



Controls



Electrical



Systems



Tools



Proceedings of the

9th INTERNATIONAL MODELICA CONFERENCE

September 3-5, 2012
Munich, Germany
www.modelica.org

Editors:

Martin Otter
Dirk Zimmer


MODELICA



DLR

Deutsches Zentrum
für Luft- und Raumfahrt e.V.
in der Helmholtz-Gemeinschaft
Robotics and Mechatronics Center



Fluids



Mechanical



Proceedings of the 9th International Modelica Conference
Munich, Germany, September 3-5, 2012

Editors:

Prof. Dr.-Ing. [Martin Otter](#) and Dr. [Dirk Zimmer](#) (DLR-RMC-SR)

Published by:

Modelica Association and Linköping University Electronic Press

ISBN: 978-91-7519-826-2

Series: Linköping Electronic Conference Proceedings, No. 76

ISSN: 1650-3686

eISSN: 1650-3740

DOI: <http://dx.doi.org/10.3384/ecp12076>

Organized by:

[Modelica Association](#)
c/o PELAB, IDA, Linköpings
Universitet
S-58183 Linköping
Sweden

and

[German Aerospace Center](#) (DLR)
Robotics and Mechatronic Center (RMC)
Institute for System Dynamics and Control (SR)
D-82234 Wessling
Germany

Conference location:

[Veranstaltungsforum Fürstenfeld](#),
Fürstenfeld 12
D-82256 Fürstenfeldbruck
Germany

Copyright © Modelica Association, 2012

Preface

The [9th International Modelica Conference](#) is the main event for users, library developers, tool vendors and language designers to share their knowledge and learn about the latest scientific and industrial progress related to Modelica, to the Modelica Association and to the [Functional Mockup Interface](#). Highlights of the conference:

- **80 regular** papers, **22 poster** papers, and **6 libraries** for the Modelica Library Award.
- **2 Keynotes**.
- **8 tutorials** (3.5 hours each, [descriptions](#)).
- **10 vendor sessions** (45 min. each) where the latest news of Modelica and FMI tools are presented.
- **17 exhibitors** in the exhibition area.

Please note that to some of the papers a Modelica library or model is attached. These files are accessible in the electronic proceedings.

The conference provides also the most important news from the Modelica Association:

- The new version of the Modelica language version 3.3 was released on May 9, 2012. There are several papers and a tutorial at the conference that discusses and demonstrates the new features.
- The working process of the Modelica Association has been changed and the work is now organized in Modelica Association Projects (MAP) with an extended board. More details are given in the presentation “Modelica News” on Tuesday, Sept.4, 9:10 – 9:25.
- The further development of the FMI (Functional Mockup Interface) standard is performed in a MAP. A draft version of FMI 2.0 will be available before the conference. An overview of this new version is given in a conference paper. In two sessions, applications and tool support for FMI are presented and discussed.
- Since July, the Modelica Association provides an open source FMI compliance checker for FMI 1.0 at https://svn.fmi-standard.org/fmi/trunk/Test_FMUs. Its purpose is to check exported FMUs for validity. The checker can also produce reference simulation results with a fixed step explicit Euler method. Shortly after FMI 2.0 is released, the compliance checker will also be available for FMI 2.0.

Finally, we want to acknowledge the support we received from the program board and program committee. We are grateful for the help by the Modelica Association and Monika Klauer from DLR. Last but not least, let us thank all authors for their contributions to these proceedings. We wish all participants an enjoyable and successful conference.

Weßling, July 20, 2012

Martin Otter and Dirk Zimmer

Organizing Committees

Program Chairs

Prof. Dr.-Ing. Martin Otter, DLR-RMC-SR, Weßling, Germany
Dr. Dirk Zimmer, DLR-RMC-SR, Weßling, Germany

Program Board

Dr. Hilding Elmqvist, Dassault Systèmes, Lund, Sweden
Prof. Peter Fritzson, Linköping University, Sweden
Dr. Michael Tiller, Dassault Systèmes, Paris, France

Program Committee

Dr. Johan Åkesson, Modelon AB, Lund, Sweden
Dr. Peter Aronsson, MathCore - A Wolfram Company, Linköping, Sweden
Prof. Karl-Erik Årzén, Lund University, Lund, Sweden
Prof. Bernhard Bachmann, Univ. Applied Sciences Bielefeld, Bielefeld, Germany
Daniel Bouskela, EDF R&D, Paris, France
Dr. David Broman, UC Berkeley, California, USA
Dr. John Batteh, Emmeskay, Michigan, USA
Dr. Ingrid Bausch-Gall, BAUSCH-GALL GmbH, Munich, Germany
Prof. Francesco Casella, Politecnico di Milano, Milano, Italy
Prof. François E. Cellier, ETH Zürich, Zürich, Switzerland
Prof. Liping Chen, Huazhong University of Science and Technology, Wuhan, China
Dr. Christoph Clauß, Fraunhofer IIS EAS, Dresden, Germany
Mike Dempsey, Claytex Services Ltd, UK
Prof. Gianni Ferretti, Politecnico di Milano, Italy
Dr. Rui Gao, Dassault Systèmes Japan, Tokyo, Japan
Anton Haumer, Technical consultant, St. Andrae-Woerden, Austria
Prof. Alberto Leva, Politecnico di Milano, Italy
Kilian Link, Siemens AG, Erlangen, Germany
Dr. Sven-Erik Mattsson, Dassault Systèmes, Lund, Sweden
Dr. Jakob Mauss, QTronic GmbH, Berlin, Germany
Ramine Nikoukhah, Altair Development France, Antony, France
Dr. Mattias Nyberg, Scania AB, Södertälje, Sweden
Dr. Hans Olsson, Dassault Systèmes, Lund, Sweden
Prof. Chris Paredis, Georgia Institute of Technology, Atlanta, Georgia, USA
Prof. Peter Pepper, TU Berlin, Berlin, Germany
Dr. Nicolas Pernet, IFP Energies nouvelles, Rueil-Malmaison, France
Dr. Adrian Pop, Linköping University, Sweden
Prof. Gerhard Schmitz, Technical University Hamburg-Harburg, Germany
Dr. Peter Schneider, Fraunhofer IIS EAS, Dresden, Germany
Dr. Stefan-Alexander Schneider, BMW, Munich, Germany
Dr. Wilhelm Tegethoff, TLK-Thermo GmbH and TU Braunschweig, Germany
Eric Thomas, Dassault-Aviation, Paris, France
Dr. Hubertus Tummescheit, Modelon AB, Lund, Sweden
Dr. Andreas Uhlig, ITI GmbH, Dresden, Germany
Prof. Alfonso Urquía, UNED, Spain
Prof. Hans Vangheluwe, University of Antwerp, Belgium and McGill University, Canada

Local Organizers

Prof.-Dr.-Ing. Martin Otter, DLR-RMC-SR, Weßling, Germany
Dr. Dirk Zimmer, DLR-RMC-SR, Weßling, Germany
Monika Klauer, DLR-RMC-SR, Weßling, Germany

Content

Session 1A: Hybrid Modeling	15
<i>Fundamentals of Synchronous Control in Modelica</i>	15
<i>A Library for Synchronous Control Systems in Modelica</i>	27
<i>State Machines in Modelica</i>	37
<i>PNlib - An Advanced Petri Net Library for Hybrid Process Modeling</i>	47
Session 1B: Thermofluid Systems	57
<i>Simulation of Non-Newtonian Fluids using Modelica</i>	57
<i>HelmholtzMedia — A Fluid Properties Library</i>	63
<i>Object-Oriented Library of Switching Moving Boundary Models for Two-phase Flow Evaporators and Condensers</i>	71
<i>High-Speed Compressible Flow and Gas Dynamics</i>	81
Session 1C Power and Energy:	101
<i>Gas Exchange and Exhaust Condition Modeling of a Diesel Engine using the Engine Dynamics Library</i>	101
<i>Library for First-Principle Models of Proton Exchange Membrane Fuel Cells in Modelica</i>	115
<i>The Modeling of Energy Flows in Railway Networks using XML-Infrastructure Data</i>	125
<i>Implementation of a Modelica Library for Energy Management based on Economic Models</i>	133
Session 1D: Electromagnetic Systems I	143
<i>Modeling and Simulation of a Linear Piezoelectric Stepper Motor in MapleSim</i>	143
<i>Magnetic Hysteresis Models for Modelica</i>	151
<i>Motor Management of Permanent Magnet Synchronous Machines</i>	159
<i>An approach for modelling quasi-stationary magnetic circuits</i>	167
Session 2A: FMI Standard I	173
<i>Functional Mockup Interface 2.0: The Standard</i>	173
<i>Generation of Sparse Jacobians for the Function Mock-Up Interface 2.0</i>	185
<i>Designing models for online use with Modelica and FMI</i>	197
<i>Co-simulation with communication step size control in an FMI compatible master algorithm</i>	205
Session 2B: Numerical Methods	215
<i>On the Formulation of Steady-State Initialization Problems in Object-Oriented Models of Closed Thermo-Hydraulic Systems</i>	215
<i>Probability-One Homotopy for Robust Initialization of Differential-Algebraic Equations</i>	223
<i>Simulating Modelica models with a Stand-Alone Quantized State Systems Solver</i>	237
<i>Fast Simulation of Fluid Models with Colored Jacobians</i>	247

Session 2C: Climate Systems I.....	253
<i>Modelling and Calibration of a Thermal Model for an Automotive Cabin using HumanComfort Library</i>	253
<i>Holistic vehicle simulation using Modelica –An application on thermal management and operation strategy for electrified vehicles.....</i>	263
<i>Modelling of Radiative Heat Transfer in Modelica with a Mobile Solar Radiation Model and a View Factor Model.....</i>	271
<i>VEPZO – Velocity Propagating Zonal Model for the prediction of airflow pattern and temperature distribution in enclosed spaces.....</i>	279
Session 2D: Mechanic Systems I.....	287
<i>Modeling and Testing of the Hydro-Mechanical Synchronization System for a Double Clutch Transmission.....</i>	287
<i>Predicting the launch feel of automatic and dual clutch transmissions.....</i>	295
<i>Modelling of Elastic Gearboxes Using a Generalized Gear Contact Model.....</i>	303
<i>Revised and Improved Implementation of the Spur Involute Gear Dynamical Model.....</i>	311
Session 3A: Mixed Simulation Techniques I.....	323
<i>Accessing External Data on Local Media and Remote Servers Using a Highly Optimized File Reader Library.....</i>	323
<i>Detailed geometrical information of aircraft fuel tanks incorporated into fuel system simulation models </i>	333
<i>Simulation of Artificial Intelligence Agents using Modelica and the DLR Visualization Library.....</i>	339
Session 3B: Embedded and Real-Time Systems.....	347
<i>Functional Development with Modelica.....</i>	347
<i>Translating Modelica to HDL: An Automated Design Flow for FPGA-based Real-Time Hardware-in-the- Loop Simulations.....</i>	355
<i>A Modelica Library for Real-Time Coordination Modeling.....</i>	365
Session 3C: Language and Compilation Concepts I.....	375
<i>Implementation of a Graphical Modelica Editor with Preserved Source Code Formatting.....</i>	375
<i>Model-based Requirement Verification : A Case Study.....</i>	385
<i>A Data-Parallel Algorithmic Modelica Extension for Efficient Execution on Multi-Core Platforms.....</i>	393
Session 3D: Mechanic Systems II.....	405
<i>Modelling and Simulation of the Coupled Rigid-flexible Multibody Systems in MWorks.....</i>	405
<i>A Modelica Library of Anisotropic Flexible Beam Structures for the Simulation of Composite Rotor Blades.....</i>	417
<i>Modeling and Simulation of a Fault-Tolerant Electromechanical Actuation System for Helicopter Swashplates in Modelica.....</i>	425

Session 4A: Language and Compilation Concepts II	433
<i>Survey of appropriate matching algorithms for large scale systems of differential algebraic equations..</i>	<i>433</i>
<i>Static and Dynamic Debugging of Modelica Models</i>	<i>443</i>
Session 4B: Control	455
<i>A Modelica Sub- and Superset for Safety-Relevant Control Applications</i>	<i>455</i>
<i>A Modelica Library for Industrial Control Systems</i>	<i>477</i>
Session 4C: Handling Simulation Output	485
<i>Modelica3D - Platform Independent Simulation Visualization.....</i>	<i>485</i>
<i>Proposal for a Standard Time Series File Format in HDF5.....</i>	<i>495</i>
Session 4D: Electromagnetic Systems II.....	507
<i>Towards a Memristor Model Library in Modelica.....</i>	<i>507</i>
<i>Fault Detection of Power Electronic Circuit using Wavelet Analysis in Modelica</i>	<i>513</i>
Session 5A: Simulation Tools	523
<i>PySimulator – A Simulation and Analysis Environment in Python with Plugin Infrastructure</i>	<i>523</i>
<i>An OpenModelica Python Interface and its use in PySimulator</i>	<i>537</i>
<i>WebMWorks: A General Web-Based Modeling and Simulation Envi-ronment for Modelica</i>	<i>549</i>
Session 5B: Mixed Simulation Techniques II.....	557
<i>Using BCVTB for Co-Simulation between Dymola and MATLAB for Multi-Domain Investigations of Production Plants.....</i>	<i>557</i>
<i>FEM models in System Simulations using Model Order Reduction and Functional Mockup Interface</i>	<i>565</i>
<i>Using Modelica models for Driver-in-the-loop simulators</i>	<i>571</i>
Session 5C: Automotive Systems.....	579
<i>Development of New Concept Vehicles Using Modelica and Expectation to Modelica from Automotive Industries</i>	<i>579</i>
<i>A Modular Technique for Automotive System Simulation.....</i>	<i>589</i>
<i>Modeling Vehicle Drivability with Modelica and the Vehicle Dynamics Library</i>	<i>599</i>
Session 5D: Power Plants.....	609
<i>Status of ClaRaCCS: Modelling and Simulation of Coal-Fired Power Plants with CO2 Capture.....</i>	<i>609</i>
<i>Start-up Optimization of a Combined Cycle Power Plant.....</i>	<i>619</i>
<i>Modeling and Simulation of a Vertical Wind Power Plant in Dymola/Modelica</i>	<i>631</i>

Session 6A: Optimization.....	641
<i>First and second order parameter sensitivities of a metabolically and isotopically non-stationary biochemical network model.....</i>	<i>641</i>
<i>Collocation Methods for Optimization in a Modelica Environment.....</i>	<i>649</i>
<i>Parallel Multiple-Shooting and Collocation Optimization with OpenModelica.....</i>	<i>659</i>
<i>Optimization Library for Interactive Multi-Criteria Optimization Tasks.....</i>	<i>669</i>
Session 6B: Mechanic Systems III.....	681
<i>A Planar Mechanical Library for Teaching Modelica.....</i>	<i>681</i>
<i>DyMoRail: A Modelica Library for modelling railway buffers.....</i>	<i>691</i>
<i>Natural frequency analysis of Modelica powertrain models.....</i>	<i>697</i>
<i>Achieving $O(n)$ Complexity for Models from Modelica.Mechanics.Multibody.....</i>	<i>705</i>
Session 6C: Climate Systems II.....	713
<i>Modeling the discontinuous individual channel injection into fin-and-tube evaporators for residential air-conditioning.....</i>	<i>713</i>
<i>Validation and Application of the Room Model of the Modelica Buildings Library.....</i>	<i>727</i>
<i>The Indoor Climate Library and its Application to Heat and Moisture Transfer in a Vehicle Cabin.....</i>	<i>737</i>
<i>Dynamic Modelling of a Condenser/Water Heater with the ThermoSysPro Library.....</i>	<i>745</i>
Session 6D: FMI Standard II.....	759
<i>FMI implementation in LMS Virtual.Lab Motion and application to a vehicle dynamics case.....</i>	<i>759</i>
<i>Generating Functional Mockup Units from Software Specifications.....</i>	<i>765</i>
<i>Functional Mock-up Interface in Mechatronic Gearshift Simulation for Commercial Vehicles.....</i>	<i>775</i>
<i>Using Functional Mock-up Units for Nonlinear Model Predictive Control.....</i>	<i>781</i>

Poster Session.....	791
<i>Modeling a Low-temperature Compressed Air Energy Storage with Modelica</i>	<i>791</i>
<i>Natural Unit Representation in Modelica</i>	<i>801</i>
<i>Modelica Code Generation with Polymorphic Arrays and Records Used in Wind Turbine Modeling.....</i>	<i>809</i>
<i>Derivative-free Parameter Optimization of Functional Mock-up Units</i>	<i>819</i>
<i>Stochastic Simulation and Inference using Modelica.....</i>	<i>829</i>
<i>A Toolchain for Real-Time Simulation using the OpenModelica Compiler.....</i>	<i>839</i>
<i>Time varying mass and inertia in paper winding multibody simulation</i>	<i>847</i>
<i>Collaborative complex system design applied to an aircraft system.....</i>	<i>855</i>
<i>Backward simulation - A tool for designing more efficient mechatronic systems.....</i>	<i>867</i>
<i>Modelling of new vehicle suspension concept with integrated electric drive.....</i>	<i>877</i>
<i>Dynamic modeling and simulation of a multi-effect distillation plant</i>	<i>883</i>
<i>Modeling a drum motor for illustrating wearout phenomena.....</i>	<i>889</i>
<i>“Green Building” – Modelling renewable building energy systems and electric mobility concepts using Modelica.....</i>	<i>897</i>
<i>High-Fidelity Transmission Simulation for Hardware-in-the-Loop Applications</i>	<i>907</i>
<i>ADGenKinetics: An Algorithmically Differentiated Library for Biochemical Networks Modeling via Simplified Kinetics Formats</i>	<i>915</i>
<i>Variable Structure Modeling for Vehicle Refrigeration Applications.....</i>	<i>927</i>
<i>Thermal Simulation of Power-Controlled Micro-CHP Systems for Residential Buildings.....</i>	<i>935</i>
<i>Modeling of a falling film evaporator</i>	<i>941</i>
<i>Integration of Modelica models into an existing simulation software using FMI for Co-Simulation.....</i>	<i>949</i>
<i>Chemical Process Modeling in Modelica.....</i>	<i>955</i>
<i>FMI Add-on for NI VeriStand for HiL Simulation</i>	<i>963</i>
<i>Using Static Parametric Design to Support Systems Engineering of Industrial Automation Systems.....</i>	<i>971</i>

Exhibitors	981
<i>BAUSCH-GALL GmbH.....</i>	<i>981</i>
<i>CENIT.....</i>	<i>981</i>
<i>Claytex.....</i>	<i>981</i>
<i>CyDesign Labs.....</i>	<i>982</i>
<i>Dassault Systèmes</i>	<i>982</i>
<i>ITI GmbH</i>	<i>982</i>
<i>LMS International</i>	<i>983</i>
<i>MapleSoft.....</i>	<i>983</i>
<i>Modelon GmbH</i>	<i>983</i>
<i>Open Modelica</i>	<i>984</i>
<i>QTronic GmbH.....</i>	<i>984</i>
<i>Schlegel Simulation GmbH.....</i>	<i>984</i>
<i>SIMPACK AG.....</i>	<i>985</i>
<i>TLK-Thermo GmbH.....</i>	<i>985</i>
<i>Transcat PLM GmbH</i>	<i>985</i>
<i>Wolfram.....</i>	<i>986</i>
<i>XRG Simulation GmbH</i>	<i>986</i>

Author Index

Abel, Andreas	775	Donders, Stijn	759
Åkesson, Johan	173, 185, 375, 619, 649, 819	Dormido, Sebastián	71, 941
Andersson, Christian	819	Drenth, Edo	847
Andersson, Daniel	101, 737	Dumont, Elisabeth	691
Andreasson, Johan	599	Dupont, Francois	37
Anthonis, Jan	759	Dziwok, Stefan	365
Antretter, Florian	949	Eckstein, Lutz	263
Arnold, Martin	173, 205	Eichberger, Alexander	775
Asghar, Adeel	443, 537	Eiden, Joerg	609
Bachmann, Bernhard	47, 185, 247, 659	El Hefni, Baligh	745
Baharev, Ali	955	Elmegaard, Brian	713
Bäker, Bernard	897	Elmqvist, Hilding	15, 27, 37, 173
Bals, Johann	513	Elsheikh, Atiyah	915
Baltzer, Sidney	263	Enge-Rosenblatt, Olaf	889
Batteh, John	599	Erdélyi, Hunor	759
Bausch-Gall, Ingrid	495	Eriksson, Lars	659
Bayer, Christian	889	Fernández, Joaquín	237
Beitelschmidt, Michael	705	Fish, Garron	571, 697
Berenguel, Manuel	883	Floros, Xenofon	237
Bergdahl, Tove	375	Förster, Michael	641
Bergero, Federico	237	Frenkel, Jens	433, 705
Blochwitz, Torsten	173, 355, 775	Friedrich, Markus	173
Bodenmüller, Tim	339	Fritzson, Peter	385, 393, 433, 443, 537, 659, 809
Bödrich, Thomas	151	Gallardo Yances, Stephanie	247, 619
Bonilla, Javier	71	Ganeson, Anand	537
Bonvini, Marco	477	Gao, Jianbo	513
Bouskela, Daniel	745	Gaucher, Fabien	37
Bouvy, Claude	263	Gebremedhin, Mahder	393, 659
Braun, Willi	185, 247	Gedda, Sofia	819
Breitenecker, Felix	557	Gentilini, Guillaume	745
Brunnemann, Johannes	609	Gissing, Jörg	263
Budt, Marcus	791	Gödecke, Andreas	565
Burhenne, Sebastian	949	Goossens, Paul	907
Casella, Francesco	215, 443	Gottelt, Friedrich	609
Cellier, François E.	71, 237	Gräber, Manuel	781
Chen, Liping	405, 549	Griffin, John	599
Clauss, Christoph	173, 205, 507	Grün, Gunnar	271, 279
Dahl, Johan	101	Gühmann, Clemens	287
Davies, Kevin	115, 801	Günther, Felix	589
de La Calle, Alberto	883, 941	Gusev, Ilya	311
Delgado Beltran, Juan Gabriel	697	Hafner, Irene	557
Dempsey, Mike	295, 571, 697	Hajek, Manfred	417
Diehl, Stefan	819	Hamann, Peter	775

Hannemann-Tamás, Ralf	641	Liu, Qinghua	549
Hartweg, Stefan	523	Ljubijankic, Manuel	323
Hasenbein, Christoph	609	Loh, Chia Choon	365
Hauger, Svein Olav	197	Magnusson, Fredrik	649
Haumer, Anton	159	Mai, Pierre R.	455
Haynes, Comas	115	Majetta, Kristin	507
Heckmann, Andreas	125	Mallebrein, Georg	589
Heinzl, Bernhard	557	Malmheden, Martin	855
Hellerer, Matthias	339, 523	Marquardt, Wolfgang	641
Herkel, Sebastian	949	Mattias, Nyberg	385
Hillmann, Claudio	809	Mattsson, Jesper	375
Hirano, Yutaka	579	Mattsson, Sven Erik	15, 37
Hodrius, Thomas	355	Mattsson, Tobias	375
Hofmann, Tobias	347	Maurer, Werner	691
Höger, Christoph	485	Mauss, Jakob	173
Huang, Hua	287	Mehlhase, Alexandra	485, 927
Huber, Jörg	323	Mikelsons, Lars	839, 971
Isakovic, Karsten	485	Mikoleit, Beate	897
Isaksson, Pär	631	Moghadam, Afshin Hemmati	393
Jahangiri, Pooyan	57	Mühlbauer, Monika	565
Jeck, Peter	263	Müller, Dirk	57, 935
Ji, Hongchao	971	Müller, Jakob	877
Ji, Yang	513	Naumann, Uwe	641
Junghanns, Andreas	173	Neumaier, Arnold	955
Kærn, Martin Ryhl	713	Neumerkel, Dietmar	173
Kastner, Wolfgang	557	Nezhadali, Vaheed	659
Kehrer, Christian	897	Nieveler, Joerg	565
Kempf, Karl	971	Nöh, Katharina	641
Kennel, Ralph	513	Norrefeldt, Victor	271, 279, 737
Kirches, Christian	781	Nouidui, Thierry	727
Kittilsen, Pål	197	Stephane	
Knoblich, René	287	Nowoisky, Sebastian	287
Kofman, Ernesto	237	Nytsch-Geusen, Christoph	323, 485
Köllner, Christian	355	Ochel, Lennart	659
Körner, Andreas	557	Olsson, Hans	173
Kosenko, Ivan	311	Oprea, Alexandra	333
Kral, Christian	159	Otter, Martin	15, 27, 173, 495, 523
Krüger, Imke	927	Palma, Cosimo	963
Kubiak, Rick	485	Paredis, Christiaan	115, 801
Kunze, Günter	433, 705	Pathak, Arnav	271, 737
Landsiedl, Michael	557	Pazold, Matthias	949
Leva, Alberto	477	Petersson, Joel	631
Liang, Feng	385	Pfeiffer, Andreas	495, 523, 537, 669
Lichius, Thomas	263	Phalak, Kaustubh	727
Liermann, Matthias	867	Picarelli, Alessandro	571
Lind, Alexandra	619	Pohlmann, Uwe	365, 765
Lind, Ingela	333	Pop, Adrian	443
Lindholm, Petter	185		
Link, Kilian	247, 619		
Liu, Qi	549		

Prescott, William	759	Sten, Jon	375
Proß, Sabrina	47	Stinner, Sebastian	935
Provan, Gregory	829	Streblow, Rita	57
Pruckner, Alfred	877	Streit, Sebastian	125
Quincy, Jean Baptiste	855	Strobel, Michael	809
Raabe, Nick	167	Suck, Julian	365
Rädler, Jörg	323	Tegethoff, Wilhelm	781
Radon, Jan	949	Thiele, Bernhard	27, 455
Ravachol, Michel	855	Thomas, Eric	855
Reddehase, Hendrik	765	Thorade, Matthis	63
Rein, Udo	775	Thüring, André	609
Reiner, Matthias	523	Tichy, Matthias	365
Renz, Ala	609	Tillack, Jana	641
Roberts, Neil	295, 697	Tobolar, Jakub	877
Roca, Lidia	883	Tummescheit, Hubertus	631, 737
Röckemann, Jens	765	Ulbrich, Heinz	589
Rodemann, Tobias	897	Unger, René	897
Roeder, Volker	609	Vahid, Orang	143, 907
Rogovchenko, Olena	385, 537	van der Linden, Franciscus	303
Romanoni, Marco	963	Velut, Stéphane	619
Rössler, Matthias	557	Venturini, Alberto	829
Ruge, Vitalij	659	Viel, Antoine	173
Saadat, Ali	63	Vittorias, Iason	565
Sadeghi, Sara	385	von Lieres, Eric	641
Sällberg, Elin	619	Vontz, Thomas	565
Samlaus, Roland	809	Wagner, Robert	765
Schäfer, Wilhelm	765	Wasbø, Stein	197
Schamai, Wladimir	385	Wellner, Kai	609
Scharff, Dirk	781	Wetter, Michael	727
Schaub, Alexander	339	Wiechert, Wolfgang	641
Schierz, Tom	205	Wischhusen, Stefan	253
Schlabe, Daniel	133	Wolf, Boris	365
Schlegel, Clemens	425	Wolf, Daniel	791
Schmidt, Torsten	507	Worschech, Niklas	839
Schmitz, Gerhard	609, 927	Wright, Derek	143
Schmitz, Moritz	641	Wyes, Jutta	641
Schneider, Stefan-Alexander	347, 455	Xie, Gang	405
Schnüttgen, Joachim	889	Xiong, Tifan	549
Schramm, Dieter	971	Yebra, Luis J.	71, 883, 941
Schubert, Christian	705	Ylikiiskilä, Johan	631
Schulze, Christian	609	Zhao, Yan	405
Schwan, Torsten	897	Zhou, Fanli	405
Seemann, Sebastian	425	Zimmer, Dirk	133, 681
Sielemann, Michael	81, 223	Ziske, Johannes	151
Sivertsson, Martin	659	Zuga, Adam	809
Sjölund, Martin	443	Zuo, Wangda	727
Soroka, Orysia	143		
Span, Roland	791		
Spieß, Christian	417		
Stavåker, Kristian	393		

Fundamentals of Synchronous Control in Modelica

Hilding Elmqvist¹ Martin Otter² Sven Erik Mattsson¹

¹Dassault Systèmes AB, Ideon Science Park, SE-223 70 Lund, Sweden

²DLR Institute of System Dynamics and Control, D-82234 Wessling, Germany

Hilding.Elmqvist@3ds.com Martin.Otter@dlr.de SvenErik.Mattsson@3ds.com

Abstract

The scope of Modelica 3.3 has been extended from a language primarily intended for physical systems modeling to modeling of complete systems by allowing the modeling of control systems and enabling automatic code generation for embedded systems.

This paper describes the fundamental synchronous language primitives introduced for increased correctness of control systems implementation. The approach is based on associating clocks to the variable types. Special operators are needed when accessing variables of another clock. This enables clock inference and increased correctness of the code since many more checks can be done during translation.

Keywords: Modelica; Synchronous; Control; Sampled Data Systems, Periodic Systems

1 Introduction

The scope of Modelica has been extended from a language primarily intended for physical systems modeling to modeling of complete systems by allowing the modeling of control systems and by enabling automatic code generation for embedded systems.

This paper describes the fundamental synchronous language primitives introduced for increased correctness of control systems implementation since many more checks can be done at compile time. A companion paper (*Elmqvist, et.al, 2012*) describes the state machine features of Modelica 3.3. Yet another companion paper (*Otter, et.al, 2012*) describes a Modelica library, *Modelica_Synchronous*, which supports a graphically oriented approach to synchronous control systems implementation.

The new language elements follow the synchronous approach (*Benveniste et. al. 2002*). They are based on the clock calculus and inference system proposed by (*Colaco and Pouzet 2003*) and implemented in Lucid Synchrone version 2 and 3 (*Pouzet 2006*). However, the Modelica approach also uses multi-rate periodic clocks based on rational arithmetic introduced by (*Forget et. al. 2008*), as an exten-

sion of the Lucid Synchrone semantics. Additionally, the built-in operators introduced in Modelica 3.3 also support non-periodic and event based clocks.

In the following sections the new language elements are discussed. Afterwards, in section 5, a rationale is given why they have been introduced by comparing the new possibilities with the features of Modelica 3.2 to model sampled data systems.

2 Synchronous Features of Modelica

The synchronous features of Modelica 3.3 will be gradually introduced by means of examples illustrating how to use them. This paper uses a completely textual approach. The companion paper (*Otter, et.al, 2012*) describes a Modelica library, *Modelica_Synchronous*, which supports a graphically oriented approach to synchronous control systems implementation.

2.1 Plant and Controller Partitioning

We will consider control of a mass and spring-damper system with a force actuator. A Modelica model is shown below:

```
model MassWithSpringDamper
  parameter Modelica.SIunits.Mass m=1;
  parameter Modelica.SIunits.TranslationalSpringConstant k=1;
  parameter
    Modelica.SIunits.TranslationalDampingConstant d=0.1;
  Modelica.SIunits.Position x(start=1, fixed=true) "Position";
  Modelica.SIunits.Velocity v(start=0, fixed=true) "Velocity";
  Modelica.SIunits.Force f "Force";
equation
  der(x) = v;
  m*der(v) = f - k*x - d*v;
end MassWithSpringDamper;
```

A simple discrete-time speed controller can be implemented as follows:

```
model SpeedControl
  extends MassWithSpringDamper;
  parameter Real K = 20 "Gain of speed P controller";
  parameter Modelica.SIunits.Velocity vref = 100 "Speed ref.";
  discrete Real vd;
```

```

discrete Real u(start=0);
equation
// speed sensor
vd = sample(v, Clock(0.01));

// P controller for speed
u = K*(vref-vd);

// force actuator
f = hold(u);
end SpeedControl;

```

The SpeedControl model extends the continuous-time plant model MassWithSpringDamper. The speed controller is a discrete-time controller. The boundaries between continuous-time equations and discrete-time equations are defined by the operators sample and hold.

The sample operator samples a continuous-time variable and returns a discrete-time variable. The sample rate is specified by the second Clock argument to sample. In this case, a periodic clock which ticks with a period of 0.01 second is specified.

Since sample returns a discrete-time result that is associated to clock Clock(0.01), the variable vd becomes discrete-time and is associated to the same clock as well. Variable vd appears in equation $u = K*(vref-vd)$ and therefore all time varying variables in this equation, i.e., u, must be also discrete-time and associated to the same clock. If further equations would be present, then all equations in which vd and u appear, would be again associated to the same clock. This approach to identify the equations belonging to the same clock is called *clock inference* and is a key element in the new approach.

The hold operator converts from discrete-time to continuous-time by holding the value between the clock ticks. More precisely, the hold(u) operator returns the start value of u if the operator is called before the first tick of the clock of u. Otherwise, the most recently available value of u is returned.

To summarize, the sample(v..) and hold(..) operators define the boundaries between clocked and continuous-time partitions. Equations and variables belonging to the same clocked partition are identified by clock inference.

2.2 Discrete-time State Variables

More advanced features will be introduced using a position controller using an inner P controller and an outer PI controller. The first version is using one clock:

```

model ControlledMassBasic
extends MassWithSpringDamper;
parameter Real KOuter = 10 "Gain of position PI controller";
parameter Real KInner = 20 "Gain of speed P controller";
parameter Real Ti = 10 "Integral time for pos. PI controller";

```

```

parameter Real xref = 10 "Position reference";

discrete Real xd;
discrete Real eOuter;
discrete Real intE(start=0);
discrete Real uOuter;

discrete Real vd;
discrete Real vref;
discrete Real uInner(start=0);
equation
// position sensor
xd = sample(x, Clock(0.01));

// outer PI controller for position
eOuter = xref-xd;
intE = previous(intE) + eOuter;
uOuter = KOuter*(eOuter + intE/Ti);

// speed sensor
vd = sample(v);

// inner P controller for speed
vref = uOuter;
uInner = KInner*(vref-vd);

// force actuator
f = hold(uInner);
end ControlledMassBasic;

```

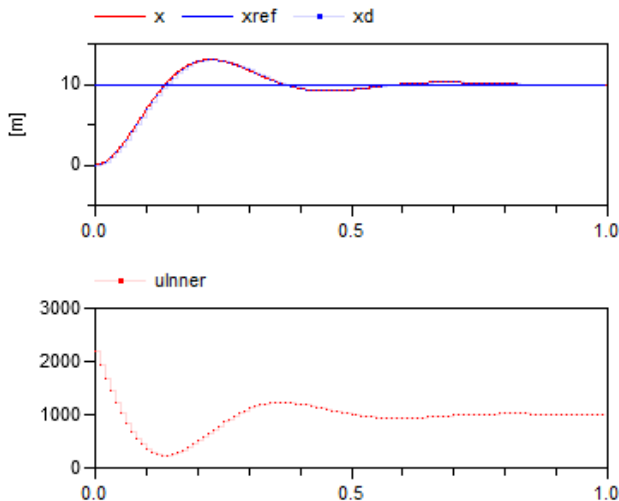
In this model, the sample operator for v does not have an associated Clock specification since it is inferred (sample(v) is implicitly associated to clock Clock(0.01) because xd is on this clock, and therefore eOuter, and therefore uOuter, and therefore vref and therefore vd, and therefore sample(v)).

Since a PI controller is used, it is necessary to introduce a discrete-time state variable for the integral part. The operator previous(.) is used to access the value of intE at the previous clock tick. Note that due to this use of previous(..), intE becomes a discrete-time state and needs to have a start value specified in the declaration (at the first clock tick, previous(intE) returns the start value of intE).

The behavior of the system is shown in the figure below: x, xref and xd (upper diagram) and the actuator signal uInner (lower diagram).

2.3 Base-clocks and Sub-clocks

A Modelica model will typically have several controllers for different parts of the plant. Such controllers might not need synchronization and can have different *base clocks*. Equations belonging to different base clocks can be implemented by asynchronous tasks of the used operating system.



It is also possible to introduce sub-clocks that tick a certain factor slower than the base clock. Such sub-clocks are perfectly synchronized with the base clock, i.e. the definitions and uses of a variable are sorted in such a way that when sub-clocks are activated at the same clock tick, then the definition is evaluated before all the uses.

Such sub-clocks can, for example, be used to save CPU resources. In some cases, an outer controller of a cascade control architecture does not need to be evaluated as often as the inner controller.

When using several clocks, it is convenient and clear to declare them. Modelica 3.3 introduces a new base type, `Clock`, for this purpose:

```
Clock cControl = Clock(0.01);
Clock cOuter = subSample(cControl, 5);
```

The `subSample` operator creates a clock which is a factor slower; in this case `cOuter` becomes 5 times slower than `cControl`. The `subSample` operator can also operate on a discrete-time variable and then picks the value at every factor clock tick of the clock of this variable.

Such clock variables can then be used as argument to the sample operator:

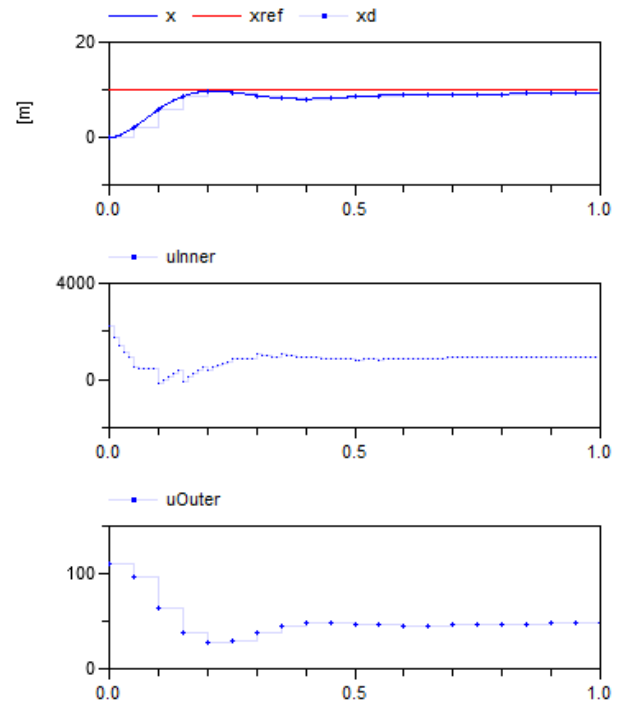
```
xd = sample(x, cOuter);
vd = sample(v, cControl);
```

The outer controller now calculates u_{Outer} at the rate of `cOuter`. u_{Outer} is the velocity reference, v_{ref} , for the inner controller which is compared to the sampled velocity measurement, vd . vd has clock `cControl`, i.e., 5 times faster than u_{Outer} . Trying to directly calculate $u_{Outer}-vd$ would give a clocking error since the semantics is not clear. The user needs to state the intent by using a clock conversion operator. In this case u_{Outer} needs to be converted to the faster clock by using the `superSample` operator:

```
vref = superSample(uOuter, 5);
```

`superSample` replicates a factor 5 times the value of the variable with the slower clock to have a clock a factor faster.

The simulation results are shown below. Note that xd and u_{Outer} have a slower sample rate than u_{Inner} .



2.4 Interval of Clock

It is possible to inquire the actual interval of a clock by using the `interval()` operator. One example of the need is when using difference approximations. Assume that no speed sensor is available and the speed needs to be estimated from changes of position. A first order approximation is shown below. It uses a faster sampling of the position, x :

```
Clock cFast = superSample(cControl, 2);
```

```
xdFast = sample(x, cFast);
vd = subSample((xdFast-previous(xdFast))/interval(), 2);
```

After approximating the derivative at the higher rate, the result is sub-sampled with a factor of 2 to get the required rate of vd .

2.5 Phase of Clock

To better control the scheduling of calculations, it is possible to shift the phase of a clock. For example, the calculation of the outer controller code will be done before the inner controller code due to the data flow. This might give jittering in the actuator signal u_{Inner} caused by the slight delay due to the computation time. One way to avoid this is to schedule the

outer code to be executed later in the cycle and to accept the use of an old value of u_{Outer} . This is accomplished in the following way:

```
Clock cOuter = subSample(shiftSample(cControl, 2, 3), 5);
```

The `shiftSample` operator shifts the clock a part of the interval. In this case $2/3$ of the interval of the clock `cControl`.

By changing the clock `cOuter` in this way, the calculation of u_{Outer} will be delayed and will not be synchronized to vd . This needs to be compensated by using `backSample` which shifts the clock in the opposite direction to `shiftSample`:

```
vref = backSample(superSample(uOuter, 5), 2, 3);
```

It should be noted that this means that a start value must be given to u_{Outer} which is used before the clock of u_{Outer} has started ticking.

The complete model including all aspects discussed above is given below:

```
model ControlledMass
  extends MassWithSpringDamper;
  parameter Real KOuter = 10 "Gain of position PI controller";
  parameter Real KInner = 20 "Gain of speed P controller";
  parameter Real Ti = 10 "Integral time for pos. PI controller";
  parameter Real xref = 10 "Position reference";

  discrete Real xd;
  discrete Real eOuter;
  discrete Real intE(start=0);
  discrete Real uOuter(start=0);

  discrete Real xdFast;
  discrete Real vd;
  discrete Real vref;
  discrete Real uInner(start=0);

  Clock cControl = Clock(0.01);
  Clock cOuter = subSample(shiftSample(cControl, 2, 3), 5);
  Clock cFast = superSample(cControl, 2);
equation
  // position sensor
  xd = sample(x, cOuter);

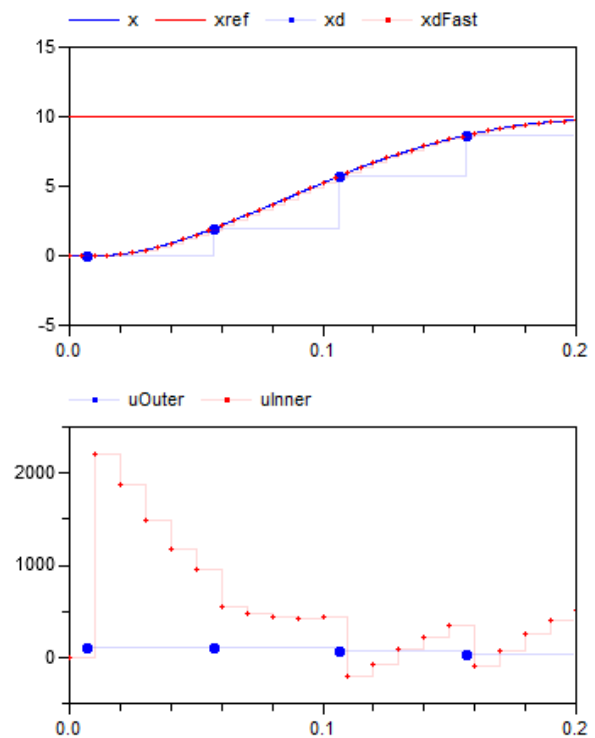
  // outer PI controller for position
  eOuter = xref-xd;
  intE = previous(intE) + eOuter;
  uOuter = KOuter*(eOuter + intE/Ti);

  // speed estimation
  xdFast = sample(x, cFast);
  vd = subSample((xdFast-previous(xdFast))/interval(), 2);

  // inner P controller for speed
  vref = backSample(superSample(uOuter, 5), 2, 3);
  uInner = KInner*(vref-vd);

  // force actuator
  f = hold(uInner);
end ControlledMass;
```

The simulation results are shown below. In particular it can be noted how u_{Outer} is shifted $2/3$ of the interval on u_{Inner} .



An interesting question is when a clock starts to tick. In principal there are two useful approaches: A clock starts ticking at time = 0 seconds or it starts ticking at the simulation start time (or when a device is switched on). The synchronous extensions of Modelica use the second approach because from the view of a hardware device, there is no absolute but only relative time.

Operator $y = \text{shiftSample}(u, c, r)$ defines a new clock that basically shifts the first activation of the clock of y in time $c/r * \text{interval}(u)$ later than the first activation of the clock of u . This definition gives not a precise time definition because $\text{interval}(u)$ is of type Real. Furthermore, it only holds in special cases, such as for periodic clocks with a fixed period. The precise time definition that holds for all clocks is achieved by constructing (conceptually) a clock `cBase`:

```
Clock cBase = subSample(superSample(u, r), c);
```

and the clock of $y = \text{shiftSample}(u, c, r)$ starts at the second clock tick of `cBase` and y is set to the most recently available value of u .

In a similar way the operator $y = \text{backSample}(u, c, r)$ defines a new clock that basically shifts the first activation of the clock of y in time $c/r * \text{interval}(u)$ before the first activation of the clock of u . Similarly to `shiftSample`, the precise time definition is achieved by constructing (conceptually) a clock `cBase`:

```
Clock cBase = subSample(superSample(u, r), c);
```

and the clock of $y = \text{backSample}(u, c, r)$ is shifted a time duration before the clock of u , such that this duration is identical to the duration between the first and second clock tick of c_{Base} .

The `backSample(..)` operator is more critical than the `shiftSample(..)` operator: The clock of v starts before the clock of u and therefore a start value for u is needed and before the first tick of the clock of u , the operator returns this start value. Additionally, there is the restriction that the clock of v cannot start before the simulation start time.

On first view, one could have only provided one operator to shift the start of a clock forward or backward in time. However, shifting backwards in time requires providing a start value, whereas this is not the case when shifting forward in time. Since these are therefore structurally different cases, it is better to use two different operators.

2.6 Exact Periodic Clocks

In the previous sections, periodic clocks are defined with the `Clock(period)` constructor, where `period` is of type `Real` and defines the sample period. The semantics is that two clocks of this kind are not time synchronized to each other. Example:

```
Clock c1 = Clock(0.1);
Clock c2 = superSample(c1,3);
Clock c3 = Clock(0.1/3);
```

Clock c_1 and c_2 are precisely time synchronized to each other and at every third tick of c_2 , clock c_1 ticks. However, clock c_3 is not time synchronized to c_1 or c_2 and there is no guarantee that c_3 ticks at every third tick of c_1 . The reason is that calculations with `Real` numbers are not exact and subject to small numerical errors.

Alternatively, a periodic clock can be defined with the `Clock(c,r)` operator, where c and r are of type `Integer`, and the fixed sample period is defined as the rational number c/r . The semantics is that all clocks defined in this way are precisely time synchronized to each other. Example:

```
Clock c1 = Clock(1,10); // period = 1/10
Clock c2 = superSample(c1,3); // period = 1/30
Clock c3 = Clock(1,30); // period = 1/30
```

Clocks c_1 , c_2 , and c_3 are precisely time synchronized to each other and at every third tick of c_2 and of c_3 , clock c_1 ticks.

An interesting question is which periods can be defined with exact periodic clocks? Basically, a period is defined as the quotient of two Modelica `Integer` numbers, which are usually 32 bit integers. Therefore, periods in the range $10^{-9} \dots 10^9$ s can be directly defined. However, clocks can be sub- and super-sampled, e.g.,

```
superSample(Clock(1, 1000000000), 1000000000);
```

The resulting clock will have a period of 10^{-18} s. In other words, from a Modelica point of view, any period that can be represented by a rational number with unlimited precision can be defined. In the Modelica 3.3 specification it is stated that “it is required that accumulated sub- and super sampling factors in the range of 1 to 2^{63} can be handled”. Therefore, every tool should support internally at least 64 bit integers and therefore periods in the range $10^{-18} \dots 10^{18}$ s.

2.7 Clocked When Clause

Although the new synchronous operators allow defining clocked equations *implicitly* due to clock inference, it is sometimes still useful to *explicitly* define that a group of equations is associated with the same clock. In order to not introduce yet another new keyword, the already existing when-clause is overloaded for this purpose. Example:

```
import Modelica.Utilities.Streams.print;
equation
when Clock(0.1) then
  x = A*previous(x) + B*u;
  y = C*previous(x) + D*u;
  print("Clock ticks at time = " + String(sample(time)));
end when
```

If a clock is used in a when-clause then all equations in the when-clause are associated with this clock. In such a case, the equations in the when-clause can be arbitrary equations (recall that for standard when-clauses with a Boolean condition, all equations in the when-clause must have a variable reference on the left hand side of every equation, i.e., equations must be of the form “ $x = \text{expr}$ ”).

In the example above, all three equations in the when-clause belong to the same partition that is associated to clock `Clock(0.1)`. When-clauses might be used to clearly define that equations are associated with the same clock. Furthermore, there are exceptional cases as in the example above, where it would be not possible to associate the `print(..)` statement to `Clock(0.1)` without a when-clause because no variable of the clocked partition is used in the print statement. If the clock of the when-clause is defined somewhere else and shall be deduced by clock inference, then the clock `Clock()` needs to be used in the when-clause:

```
when Clock() then // clock is inferred
  x = A*previous(x) + B*u;
  y = C*previous(x) + D*u;
  print("Clock ticks at time = " + String(sample(time)));
end when
```

In Modelica 3.3, clocked when-clauses are restricted: The condition must be a clock (and not, say a Boolean expression of clocks such as “ c_1 or c_2 ”), an else-

when part is not allowed, and the clocked when clause can only appear in an equation section.

2.8 Varying Interval Clocks

It is also possible to define clocks with a varying interval between the sampling points. As an example, consider

```

model VaryingClock
  Integer nextInterval(start=1);
  Clock c = Clock(nextInterval, 100);

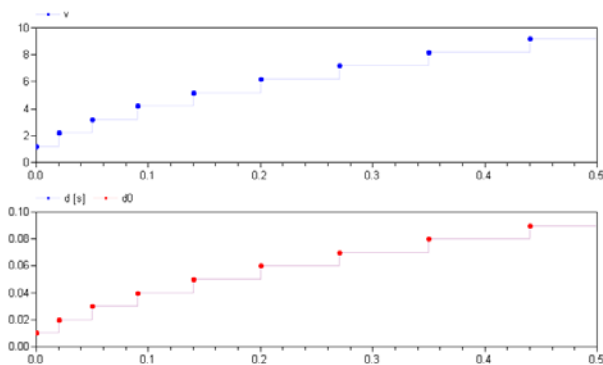
  Real v(start=0.2);
  Real d = interval(v);
  Real d0 = previous(nextInterval)/100.0;
equation
  when c then
    nextInterval = previous(nextInterval) + 1;
    v = previous(v) + 1;
  end when;
end VaryingClock;
    
```

It defines a Clock c with varying interval, nextInterval. A definition of the form

```

Clock c = Clock(nextInterval, 100)
    
```

states that clock c ticks at the simulation start and then every nextInterval/100 seconds, and at every clock tick, nextInterval can be newly computed. Since at the first clock tick, previous(nextInterval) is equal to the start value of nextInterval (= 1), the value of nextInterval at the first clock tick is 1+1 = 2, and therefore the second clock tick is at 2/100 seconds. The further ticks are at 5/100, 9/100 etc. The behavior of the variable v is shown in the following plot:

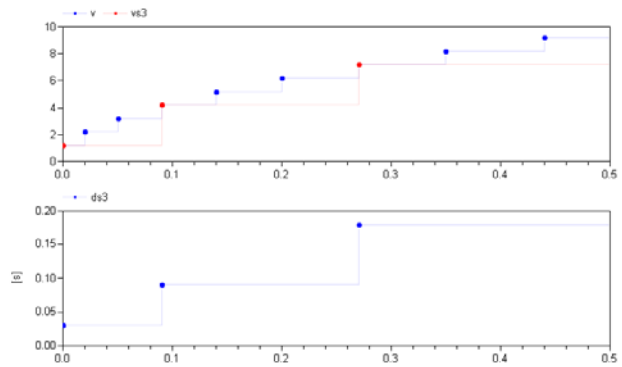


The variables $d = \text{interval}(v)$ and $d0 = \text{previous}(\text{nextInterval})/100.0$ are equal.

Let us sub-sample v by adding to the model:

```

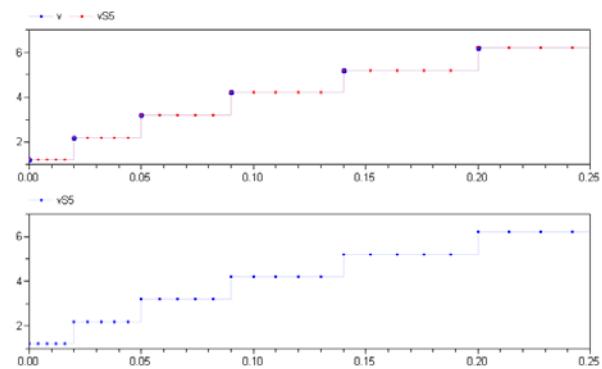
Real vs3 = subSample(v, 3);
Real ds3 = interval(vs3);
    
```



As the plot shows, vs3 samples each third point of v. We can also super-sample:

```

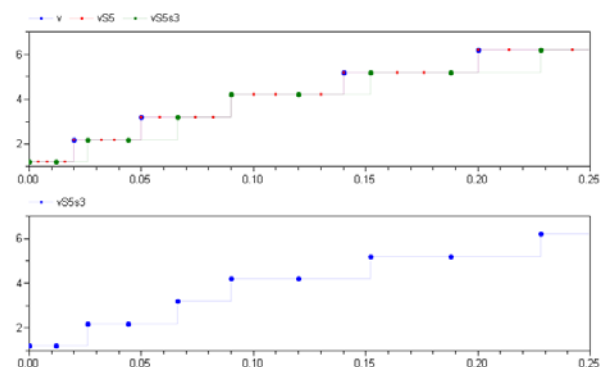
Real vS5 = superSample(v, 5);
Real dS5 = interval(vS5);
    
```



The 5 super-sampling points are evenly distributed in time within the intervals of clock c as shown by the plot of ds5. Let us now sub-sample ds5:

```

Real vS5s3 = subSample(vS5, 3);
Real ds3S5 = interval(vS5s3);
    
```



The result is that vS5s3 is every third sample of vS5 resulting in a more irregular sampling interval. The equation $vS5s3 = \text{subSample}(vS5, 3)$ can be expanded as $vS5s3 = \text{subSample}(\text{superSample}(v, 5), 3)$.

What is the result if we do it in the reverse order, $vs3S5 = \text{superSample}(\text{subSample}(v, 3), 5)$? For the clock c, the time to the next tick is known at the current tick.

However, this is not the case for the clock of `subSample(v, 3)`. The interval to its next tick is the sum of 3 future intervals of `c` and only the first term is known. The definition of super-sampling does not require the intervals of super-sampling to be equidistant in time. The definition is instead based on counting ticks. It means that `vs3S5 = vS5s3`.

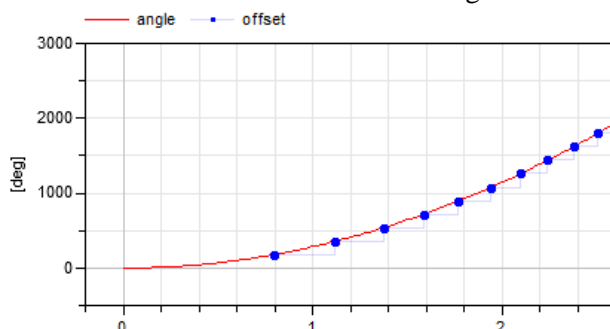
In Modelica, a non-periodic clock can only be introduced by using an explicit clock constructor. The factors of sub-sampling or super-sampling must be parameter expressions, which mean that neither sub-sampling nor super-sampling can construct a clock with varying interval from a periodic clock. It is also required that there must be only one clock constructor, `c`, in the same base-clock partition if `c` is a non-periodic clock. All this means that we can construct a new clock `c0` that is a super-sampled clock of `c`, such that all other clocks can be modeled as pure sub-sampling clocks of `c0`. As we have described, there are no issues in making a faster clock `c0` by super-sampling `c`. The sub-sampling of `c0` to implement all the sub-clocks is then just a matter of counting ticks and picking the *n*th samples.

2.9 Boolean Clocks

It is also possible to define clocks that tick when a Boolean expression changes from false to true. For example assume that a clock shall tick whenever the shaft of a drive train passes 180° . This can be defined as (Otter, et.al. 2012):

```
w = der(angle);
J*der(w) = tau;
when Clock(angle >= hold(offset)+Modelica.Constants.pi) then
  offset = sample(angle);
end when;
```

At the simulation start the discrete variable `offset` has a start value of zero. Therefore, the first clock tick appears when `angle` becomes larger as 180° . Then, `offset` is set to the actual angle, and the next clock tick appears at another full rotation of the shaft. Note, that the Boolean expression is continuous-time, and therefore the clocked variable `offset` cannot be directly used, but must be casted from a clocked to a continuous-time variable with operator `hold`. A typical simulation result is shown in the next figure:



Operators `subSample`, `superSample`, `shiftSample` and `backSample` can also be applied on Boolean clocks. However, there are restrictions. For example, `superSample(..)` cannot introduce new ticks because the next clock tick is not known in advance. Example:

```
Clock u = Clock(sine(time) > 0);
Clock y1 = subSample(u,4);
Clock y2 = superSample(y1,2); // fine y2 = subSample(u,2)
Clock y3 = superSample(u, 2); // error
```

2.10 Discretized continuous time

A partition (i.e., a set of equations) that is marked by `sample`, `hold`, `subSample`, `superSample` etc. operators is called a “clocked partitions”. There are two different kinds of clocked partitions:

Clocked discrete-time partition

This is the type of partition discussed so far, consisting of algebraic equations, potentially using operators `previous(..)` and `interval(..)` in the equations.

Clocked discretized continuous-time partition

This is a partition where the operator `der(..)` is used (and then `previous(..)` and `interval(..)` must not be present). In such a case a set of differential and algebraic equations is marked to be a clocked partition. The semantics is that at clock ticks these equations are solved with a specified integration method from the previous to the next clock tick. The integrator for such a partition is propagated (inferred) similarly as a clock and therefore it suffices to define it at a few places.

This is a powerful feature since in many cases it is no longer necessary to manually implement discrete-time components but it suffices to just build-up a controller with continuous-time components and then sample the input signals and hold the output signals.

In the following example a continuous-time PI controller that gets a reference and a measurement signal as input is automatically transformed to a clocked partition:

```
model ClockedPI
  parameter Real k;
  parameter Real T;
  input Real y_ref;
  input Real y_mes;
  output Real u(start=0.0);
  discrete Real e;
  discrete Real x;
  discrete Real ud;
  Clock c = Clock(Clock(0.1), solverMethod="ImplicitEuler");

equation
  // Sampling the inputs
  e = sample(y_ref,c) - sample(y_mes);
```

```
// PI controller
der(x) = e/T;
ud = k*(x + e);

// Holding the output
u = hold(ud);
end ClockedPI;
```

With the declaration `Clock(c, solverMethod)`, the `solverMethod` (defined as `String`) is associated to clock `c` and the partitions to which this clock is associated are solved with the specified solver method (= integration method). As already mentioned, this feature can be used to discretize continuous-time blocks. Also, nonlinear plant models can be inverted and the inverse model can be discretized and used, say, as feedforward controller part in a sampled data controller, see (Otter, et. al. 2012). Furthermore, this feature can be utilized for multi-rate real-time simulations where a model is partitioned in different parts and these parts are solved with different integration methods and step sizes.

3 Synchronous Operators

All newly introduced operators of the synchronous extension to Modelica have been sketched so far. In this section, a short overview of these operators is given:

Clock Constructors

`Clock()`: Returns a clock that is inferred

`Clock(i,r)`: Returns a variable interval clock where the next interval at the current clock tick is defined by the rational number i/r . If i is parametric, i.e., a literal, a constant, a parameter or an expression of those kinds, the clock is periodic.

`Clock(ri)`: Returns a variable interval clock where the next interval at the current clock tick is defined by the Real number ri . If ri is parametric, the clock is periodic.

`Clock(cond, ri0)`: Returns a Boolean clock that ticks whenever the condition `cond` changes from false to true. The optional `ri0` argument is the value returned by operator `interval()` at the first tick of the clock.

`Clock(c,m)`: Returns clock `c` and associates the solver method `m` to the returned clock .

Base-clock conversion operators

`sample(u,c)`: Returns continuous-time variable `u` as clocked variable that has the optional argument `c` as associated clock.

`hold(u)`: Returns the clocked variable `u` as piecewise constant continuous-time signal. Before the

first tick of the clock of `u`, the start value of `u` is returned.

Sub-clock conversion operators

`subSample(u,factor)`: Sub-samples the signal or clock `u` by the integer factor. If `factor` is not present, it is inferred.

`superSample(u,factor)`: Super-samples the signal or clock `u` by the integer factor. If `factor` is not present, it is inferred.

`shiftSample(u,c,r)`: Shifts the clock of a signal or clock `u` forward in time.

`backSample(u,c,r)`: Shifts the clock of a signal or clock `u` backward in time. Before the first tick of the clock of `u`, the start value of `u` is returned.

Other operators

`previous(u)`: At the first tick of the clock of `u`, the start value of `u` is returned. At subsequent clock ticks, the value of `u` from the previous clock activation is returned.

`interval(u)`: Returns the interval between the previous and the present tick of the clock to which signal `u` is associated. The interval is returned as a Real number.

4 Base-clock and Sub-clock Partitioning

Consider the example `SpeedControl` in section 2.1. The variables and equations of `MassWithSpringDamper` form a well-defined continuous-time model together with the equation $f = \text{hold}(u)$ from `SpeedControl` if we view `u` as a known input. Similarly the variables and equations added in `SpeedControl` when extending from `MassWithSpringDamper` form a well-defined discrete system if we disregard the equation $f = \text{hold}(u)$, which already is used in the continuous time system and if we view `v`, referred in the equation $\text{vd} = \text{sample}(v, \text{Clock}(0.01))$ as a known input. We have now decomposed the system in a *continuous-time partition* and in a *discrete-time partition*.

For the general case, we observe that the sample and hold operators serve an important role as identifying the interfaces between the two kinds of partitions. The first argument of `sample` identifies inputs to discrete-time partitions that must be provided by continuous time partitions. Similarly the first argument of `hold` identifies inputs to continuous-time partitions that must be provided by discrete-time partitions. If the first arguments are expressions, auxiliary variables are introduced.

The idea of the base-clock decomposition is to decompose the variables and the equations into sets where the equations only refer to variables of its own set if we neglect references of the first argument of sample and hold. There are simple algorithms for doing this, for details, see (*Modelica Association 2012*).

It must then be possible to classify a partition as either continuous-time or discrete-time. Use of previous, subSample, superSample, shiftSample or backSample or appearances of clocks or clock constructors requires the partition to be discrete-time. The global variable time can only be referenced in a continuous time partition.

The derivative operator is clearly a continuous-time operator. However, it may appear in a discrete-time partition, because there are features to have them automatically discretized by defining appropriate solver clocks, see section 2.10.

The discrete time partitions are further divided into sub-clock partitions by the same procedure while treating the first argument of the operators subSample, superSample, shiftSample or backSample as known inputs.

The result of sub-clock partitioning for the model ControlledMass in section 2.5 is:

Continuous-time partition:

```
der(x) = v;
m*der(v) = f - k*x - d*v;
f = hold(uInner);
```

Discrete-time sub-partition 1:

```
xd = sample(x, cOuter);
eOuter = xref-xd;
intE = previous(intE) + eOuter;
uOuter = KOuter*(eOuter + intE/Ti);
```

Discrete-time sub-partition 2:

```
xdFast = sample(x, cFast);
aux1 = (xdFast-previous(xdFast))/interval();
```

Discrete-time sub-partition 3:

```
vd = subSample(aux1, 2);
vref = backSample(aux2, 2, 3);
uInner = KInner*(vref-vd);
```

Discrete-time sub-partition 4:

```
aux2 = superSample(uOuter, 5);
```

5 Rationale for Clocked Semantics

This section describes why the synchronous language elements have been introduced in Modelica 3.3, by analyzing the issues of Modelica 3.2 regarding control systems implementation.

Modelica 3.2 has both continuous-time and discrete-time equations. Discrete-time equations are enclosed in when-clauses and are only executed at certain events, i.e. these equations are only valid in-

stantaneously, not always. Furthermore, the discrete-time equations are not general equations, since the left hand-side of an equation in a when-clause must be a variable reference. It is for example not allowed to write in a when-clause: “ $A*x = b$ ”. The synchronous features of Modelica 3.3 remove this restriction and general equations are allowed in clocked partitions and in particular also in clocked when-clauses.

In order to handle such instantaneous equations, a special semantics regarding the definition of variables was introduced. A variable that is assigned by an instantaneous equation keeps its value until the next event when it is assigned again (= automatic “hold” semantics). This implies that the value of such a discrete-time variable could be read at any time by another instantaneous equation or continuous-time equation.

Such semantics can, however, be error prone when different discrete-time equations are not correctly synchronized (see example below). The synchronous features of Modelica 3.3 remove this problem.

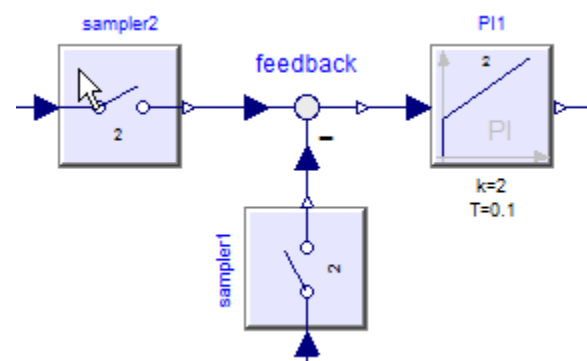
Periodically sampled control systems can be defined with standard Modelica 3.2 when-clauses and the sample operator. For example:

```
when sample(0,3) then
  xd = A*pre(xd) + B*y;
  u = C*pre(xd) + D*y;
end when;
```

This approach to define periodically sampled data systems has the following drawbacks that are not present with the solution using clocks and clocked equations described earlier in this paper:

Sampling errors cannot be detected:

All current Modelica libraries modeling sampled data systems, such as Modelica.Blocks.Discrete, or Modelica_LinearSystems2.Controller (*Baur, et.al. 2009*) provide a set of blocks where at every block instance the sample period has to be defined in some way. For example, the following figure shows part of a control system modeled with the Modelica_LinearSystems2 library:



At every discrete block (here: `sampler1`, `sampler2`, `PI1`) a `sampleFactor` has to be given defining that the block equations are sampled at a multiple of a base sampling rate (which is propagated via inner/outer to all instances). This factor is shown in the icons (here: “2”). If the modeler accidentally gives a different number at one of the blocks (e.g., at “`sampler1`”), then this is still a correct Modelica model and a translator has to accept it, although this controller is erroneous.

Furthermore note that component “feedback” is still a continuous-time model without a `when`-clause. If everything is correctly modeled, the “effect” of the above model is that of a sampled data system with one periodic sampling rate. However, it is easy to make a mistake (e.g. forgetting “`sampler1`”, or using a `sampleFactor` of 3 at one component), and then the resulting model does no longer describe the desired controller, but is still a valid Modelica model.

Worse, there is no easy way for a tool to figure out which equations belong to one partition that should be downloaded to a hardware device (e.g., describes the above figure one controller with one sample rate, or three different controllers that are connected by the continuous-time block “feedback”?). Due to the automatic sample and hold semantics of `when`-clauses in Modelica, it is not possible to fix this with Modelica 3.2 language elements.

With the synchronous language elements partitions are identified that belong to the same clock. The sampling rate has to be defined only at one place. Sampling errors can be easily detected, since then the requirement is violated somewhere that all variables in a clocked equation must belong to the same clock.

Unnecessary initial values have to be defined:

Due to the automatic sample and hold semantics, all variables assigned in a `when`-clause must have an initial value because they might be used before they are assigned a value the first time. Example:

```
when b then
  y1 = 2*x;
end when;
y2 = 2*y1;
```

Since the continuous-time equation $y2 = 2*y1$ is valid all the time, including during initialization, a value for `y1` is needed all the time. The `when`-clause in the example is not active during initialization, and therefore an initial value for `y1` has to be provided. In general, it is too difficult and probably impossible that a tool can figure out whether an initial value for a discrete-time variable in Modelica 3.2 is needed or not. The only safe way is therefore to provide initial values for all discrete-time variables, although in reali-

ty, only a small sub-set of the discrete-time variables needs an initial value.[^]

With the synchronous language elements this is different: Start values are required for the first arguments of some operators (`previous`, `hold`, `backSample`). For all other variables, it is guaranteed that a start value is not needed for initialization (it might be needed as guess value for an iteration variable of a nonlinear equation system).

Inverse models not supported in discrete systems:

It is not possible to use a continuous-time model in `when` clauses. However, this feature is highly desirable. For example, some advanced controllers use an inverse model of a plant in a controller, see (*Looye et. al. 2005*). This powerful feature of Modelica to use a nonlinear plant model in a controller is only available for continuous-time systems, but not for discrete-time systems. With Modelica 3.2, modelers therefore have to export an inverse plant model and, e.g. Dymola provides the export option to include an integration method and treat the exported component from the outside as discrete-time system. It is then possible to import this discrete-time component in another environment, but not in a Modelica model. With clocked equations of Modelica 3.3, clocked controllers with continuous-time models can be directly defined in Modelica, see section 2.10.

Efficiency degradation at event points:

Simulating a continuous-time plant and a discrete-time controller in Modelica 3.2 together results in an event iteration at a sample instant. A `when`-clause with a `sample(..)` condition is evaluated exactly once at such an event instant. However, the continuous-time model to which the sampled data controller is connected will be evaluated typically three times at a sample instant: Once, when the sample instant is reached, once to evaluate the continuous equations at the sample instant, and once when an event iteration occurs since a discrete variable `v` is changed and `pre(v)` appears in the equations. Since a sampled system is only evaluated once at a sample instant, i.e., at a particular time instant, event iteration should not be necessary since the discrete-time variables cannot be changed by the event iteration. However, it seems to be difficult to figure this out automatically for a Modelica 3.2 model and therefore Modelica tools, including Dymola, have usually at least one unnecessary evaluation of the continuous-time equations at a sample instant.

With clocked equations described in the next sections a tool does not need to trigger an event iteration, because it is guaranteed that all equations belonging to a periodic or non-periodic interval clock are evaluated exactly once at an event instant, and

variables computed in such a partition cannot be used outside of the partition (only with a cast operator the most recent available value of a clocked variable v can be inquired outside of the clocked partition, but not $\text{previous}(v)$), and therefore event iteration cannot give a different result. Therefore, it is easy for a tool to avoid the unnecessary re-evaluation of the continuous-time equations at an event triggered by a clock.

6 Conclusions

We have introduced synchronous features in Modelica. For a discrete-time variable, its clock is associated with the variable type. Special operators have to be used to convert between clocks. This gives an additional safety since correct synchronization is guaranteed by the compiler. It would have been very hard to correctly implement the last version of the example control system without such help from the compiler.

7 Acknowledgements

The authors are very thankful to Albert Benveniste, Marc Pouzet, Benoit Caillaud, Timothy Bourke, Francois Dupont, Daniel Weil, Fabien Gaucher, Torsten Blochwitz, Peter Fritzson, Hans Olsson and Modelica Association members for stimulating discussions and feedback during evolutions of the Modelica 3.3 specification.

Parts of this work were supported by the German BMBF (Förderkennzeichen: 01IS08002), and the Swedish VINNOVA (funding number: 2008-02291) within the ITEA2 MODELISAR project (<http://www.itea2.org/project/result/download/result/5533>). The authors appreciate the partial funding of this work.

References

- Baur M., Otter M., and Thiele B. (2009): **Modelica Libraries for Linear Control Systems**. Proceedings of 7th International Modelica Conference, Como, Italy, September 20-22. www.ep.liu.se/ecp/043/068/ecp09430068.pdf
- Benveniste A., Caspi P., Edwards S.A., Halbwachs N., Le Guernic P., and Simone R. (2003): **The Synchronous Languages Twelve Years Later**. Proc. of the IEEE, Vol., 91, No. 1. www.irisa.fr/distribcom/-benveniste/pub/synch_ProcIEEE_2002.pdf
- Colaco J.-L., and Pouzet M. (2003): **Clocks as First Class Abstract Types**. In Third International Conference on Embedded Software (EMSOFT'03), Philadelphia, Pennsylvania, USA, October 2003. www.di.ens.fr/~pouzet/lucid-synchrone/papers/emsoft03.ps.gz
- Elmqvist H., Gaucher F., Mattsson S.E, and Dupont F. (2012): **State Machines in Modelica**. Proceedings of 9th International Modelica Conference, Munich, Germany, September 3-5.
- Forget J., F. Boniol, D. Lesens, C. Pagetti (2008): **A Multi-Periodic Synchronous Data-Flow Language**. In 11th IEEE High Assurance Systems Engineering Symposium (HASE'08), Dec. 3-5 2008, Nanjing, China, pp. 251-260. <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?reaod=true&arnumber=4708883&contentType=Conference+Publications>
- Modelica Association (2012): **Modelica Language Specification Version 3.3**. <https://www.modelica.org/documents/ModelicaSpec33.pdf>.
- Otter M., Thiele B., and Elmqvist H. (2012): **A Library for Synchronous Control Systems in Modelica**. Proceedings of 9th International Modelica Conference, Munich, Germany, September 3-5.
- Pouzet M. (2006): **Lucid Synchrone, Version 3.0, Tutorial and Reference Manual**. <http://www.di.ens.fr/~pouzet/lucid-synchrone/>
- Looye G., Thümmel M., Kurze M., Otter M., and Bals J. (2005): **Nonlinear Inverse Models for Control**. Proceedings of 4th International Modelica Conference, ed. G. Schmitz, Hamburg, March 7-8. https://www.modelica.org/events/Conference2005/online_proceedings/Session3/Session3c3.pdf

A Library for Synchronous Control Systems in Modelica

Martin Otter¹, Bernhard Thiele¹, Hilding Elmqvist²

¹DLR Institute of System Dynamics and Control, D-82234 Wessling, Germany

²Dassault Systèmes AB, Ideon Science Park, SE-223 70 Lund, Sweden

Martin.Otter@dlr.de, Bernhard.Thiele@dlr.de, Hilding.Elmqvist@3ds.com

Abstract

Based on the synchronous language elements introduced in Modelica 3.3, a library is described to utilize the new features in a convenient way for graphical model definition of sampled data systems. The library has elements to define periodic clocks and event clocks that trigger elements to sample, sub-sample or super-sample partitions synchronously. Optionally, quantization effects, computational delay or noise can be simulated. Continuous-time equations can be automatically discretized and utilized in a sampled data system. This is demonstrated by using the inverse of a nonlinear plant model in the feed forward path of a discrete controller of a mixing unit.

Keywords: Synchronous models, sampled data systems, periodic systems, clock, inverse systems

1 Introduction

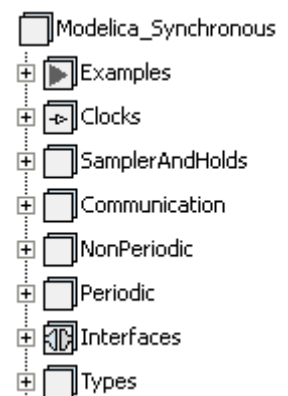
In the Modelica language version 3.3 (*Modelica Association 2012*) synchronous language features have been introduced to precisely define and synchronize sampled data systems with different sampling rates. This paper is a companion paper to (*Elmqvist et.al. 2012*) which should be first inspected to understand why new language elements have been introduced, as well as the syntax and semantics of them.

The new language elements follow the synchronous approach (*Benveniste et. al. 2002*). They are based on the clock calculus and inference system proposed by (*Colaco and Pouzet 2003*) and implemented in Lucid Synchrone version 2 and 3 (*Pouzet 2006*). However, the Modelica approach also uses multi-rate periodic clocks based on rational arithmetic introduced by (*Forget et. al. 2008*), as an extension of the Lucid Synchrone semantics. Additionally, the built-in operators of Modelica 3.3 also support non-periodic and event based clocks¹.

In order to utilize these elements in an actual model in a *convenient way*, a free library “Modelica_Synchronous” has been developed using a prototype of Dymola (*Dassault Systèmes 2012*) for the

new language elements. This library is in a prototype status. After an evaluation period it is planned to include this library into the Modelica Standard Library. Note, all Modelica libraries designed so far for sampled systems, such as Modelica.Blocks.Discrete, Modelica_LinearSystems2.Controller (*Baur et. al. 2009*) and Modelica_EmbeddedSystems (*Elmqvist et.al. 2009*) are becoming obsolete and should be replaced by this new library.

In the figure to the right a screenshot of the library is shown with the first sub-library level. The most important sub libraries are:



- **Clocks:**
Library of blocks that generate clocks.
- **SamplerAndHolds:**
Library of blocks that sample, sub-sample, super-sample and hold signals.
- **NonPeriodic:**
Library of blocks that operate on periodically and non-periodically clocked signals (the blocks depend explicitly on the *actual* sample interval).
- **Periodic:**
Library of blocks that are designed to operate only on periodically clocked signals, mainly described by z transforms (the blocks do not *explicitly* depend on the sample period, but implicitly, since the block parameters need to be designed for one specific sample period).

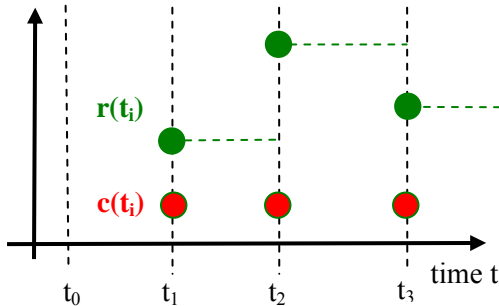
In the following subsections, the most important blocks are discussed and their usage demonstrated in examples.

2 Clocks

A “Clock” is a new base data type introduced in Modelica 3.3 (additionally to Real, Integer, Boolean, String) that defines when a particular partition of equations of a model is active. Every variable and every equation is either continuous-time or is associ-

¹ A non-periodic clock is defined by a varying interval and an event clock by a Boolean condition.

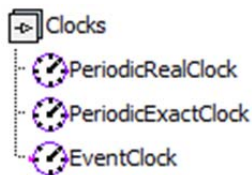
ated exactly to one clock (*Elmqvist et.al. 2012*). This feature is visualized in the figure below where $c(t_i)$ is a clock that is active at particular time instants and $r(t_i)$ is a variable that is associated to this clock. A clocked variable has only a value when the corresponding clock is active:



Similarly to RealInput, RealOutput etc., clock input and output connectors are defined in sub library “Interfaces” in order to propagate clocks via connections:

Icon	Modelica Definition
	<code>connector ClockInput = input Clock;</code>
	<code>connector ClockOutput = output Clock;</code>

Sub library “Clocks”, see screenshot to the right, defines the following components that generate clocks, and provide the respective clock via its ClockOutput connector to other components:



- **PeriodicRealClock** defines a periodic clock where the period is defined with a Real number (e.g. “period = 0.1” for 0.1 s). If clocks are related relatively to each other (see section 4), then only one of them can be a PeriodicRealClock.
- **PeriodicExactClock** defines a periodic clock with a resolution defined by enumeration “Types.Resolution” (with values “y, d, h, min, s, ms, us, ns”) and an integer multiple “factor” of this resolution. For example “factor = 3” and “resolution = Types.Resolution.ms” defines a periodic clock with sample period 3 ms.
- **EventClock** defines a clock that is active when the Boolean input to this component changes from false to true.

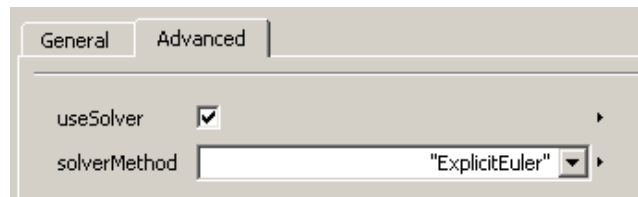
The implementation of these clocks is a direct mapping to the new clock generators. Example:

```
block PeriodicRealClock
  parameter Modelica.SIunits.Time period;
  extends Modelica_Synchronous.Interfaces.PartialClock;
```

```
equation
  y = Clock(period);
end PeriodicRealClock;

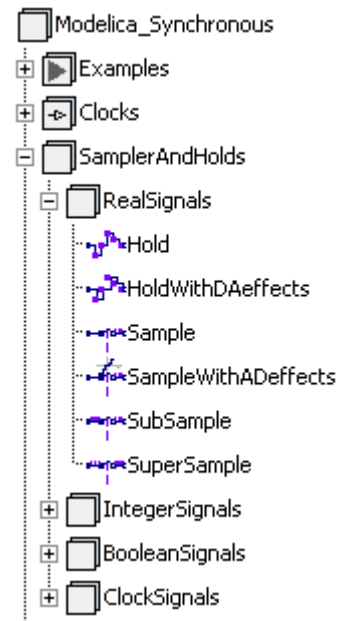
partial block PartialClock
  parameter Boolean useSolver = true
  annotation(Dialog(tab="Advanced"));
  parameter Modelica_Synchronous.Types.SolverMethod
  solverMethod="External"
  annotation(Dialog(tab="Advanced",enable=useSolver));
  Modelica_Synchronous.Interfaces.ClockOutput y;
end PartialClock;
```

All these clocks have an “Advanced” menu in which an optional integration method (such as “explicit Euler method”) can be associated to the clock, see next figure. The effect of such a definition will be explained below.



3 Sample and Hold

Within the sub library “SamplerAndHolds” various blocks are defined to sample, sub-sample, super-sample and hold signals. Since Modelica does not have generic types, for every base type a separate sub-library is present, such as SamplerAndHolds.RealSignals, see screenshot to the right. All these components define boundaries between different partitions, especially:



- **Sample** requires that the input signal is continuous-time. The block samples the input and provides it as clocked output signal. The equations that have a dependency to that output, are collected/grouped into the same clocked partition.
- **Hold** requires that the input signal is clocked and provides it as continuous-time signal to the output with a zero order hold. Before the first tick of the clock that is associated to the input, the output is set to parameter y_start (this value is also displayed in the icon, see Figure 1).

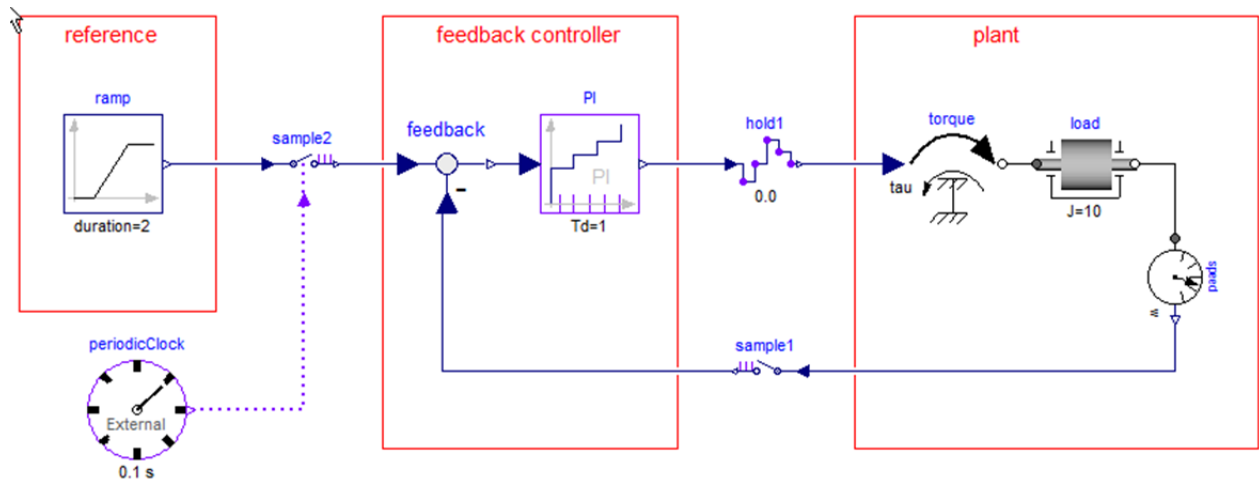


Figure 1: Simple drive train with clocked PI controller, samplers, hold and periodic clock.

- **SampleWithADeffects**, **HoldWithDAeffects** are similar to Sample and Hold, but provide additionally the options to simulate particular effects, such as noise, signal limitations and quantization effects, as well as computational delays.

The Sample and Hold blocks have again a direct mapping to the corresponding new language elements. For example, the RealSignals.Sample block is implemented as:

```

block Sample
  parameter Boolean useClock=false;
  Modelica.Blocks.Interfaces.RealInput u;
  Modelica.Blocks.Interfaces.RealOutput y;
  Modelica_Synchronous.Interfaces.ClockInput
  clock if useClock;
protected
  Modelica_Synchronous.Interfaces.ClockInput c_internal;
equation
  connect(clock, c_internal);
  if useClock then
    y = sample(u,c_internal);
  else
    y = sample(u);
  end if;
end Sample;

```

With the default option `useClock=false`, just the input `u` is sampled, $y = \text{sample}(u)$, and the clock of the output `y` is deduced by clock inference due to the clock definition somewhere else (Elmqvist et al. 2012).

If `useClock=true`, the input clock connector clock is enabled and the clock propagated to this connector is used as clock for the output: $y = \text{sample}(u, \text{clock})$, see block `sample2` in Figure 1.

Figure 1 demonstrates all blocks that have been discussed so far within an illustrative example model. This model consists of a load inertia that is driven by a torque. The goal is to control the speed of the

inertia. For this, a feedback controller is provided in form of a periodic sampled data system described with clocked equations. The reference part is again a continuous-time model and provides the desired speed of the inertia.

The boundaries of the feedback controller are defined with components `sample1`, `sample2` and `hold1` that are instances of blocks `Sample` and `Hold` respectively. All equations inside this partition (“feedback controller”) need to be associated to a clock. For this, the `Sample` block has an optional `ClockInput` connector that can be enabled. In the figure, a periodic clock with period 0.1 s is connected to `sample2` and therefore the “feedback controller” partition is active every 0.1 s. Note, it would also be fine to connect the clock additionally to `sample1`, since associating the same clock definition several times to a partition is allowed.

The PI component is a clocked block from `Modelica_Synchronous.NonPeriodic`. It is implemented as (note, `previous(x)` defines that `x` is clocked and that the value from the previous clock tick is used; `interval(u)` is the time duration from the previous to the actual clock tick as Real number):

```

block PI "From Modelica_Synchronous.NonPeriodic"
  extends Modelica_Synchronous.Interfaces.PartialClockedSISO;
  parameter Real k "Gain of continuous PI controller";
  parameter Real T "Time constant of continuous PI controller";
  output Real x(start=0) "Discrete PI state";
protected
  Real Ts = interval(u) "Sample period";
equation
  x = previous(x) + u*Ts/T;
  y = k*(x + u);
end PI;

```

This PI controller is parameterized with the coefficients of a continuous-time PI controller and with the actual sample period the coefficients of the discretized (clocked) PI controller are computed. Changing

the sample period will therefore result in a similar controller behavior.

It would also be possible to utilize the PI controller from the Modelica_Synchronous.Periodic sub-library. In this sub-library it is assumed that the blocks are utilized only with periodic clocks and the block parameters have been designed for a particular sample period. The corresponding PI controller is implemented as:

```

block PI "From Modelica_Synchronous.Periodic"
  extends Modelica_Synchronous.Interfaces.
    PartialPeriodicallyClockedSISO;
  parameter Real kd "Gain of discrete PI controller";
  parameter Real Td "Time constant of discrete PI controller";
  output Real x(start=0) "Discrete PI state";
equation
  x = previous(x) + u/Td;
  y = kd*(x + u);
end PI;
    
```

The PI coefficients k_d and T_d are designed for a particular sample period. Changing this sample period, without changing k_d and T_d , will significantly change the controller behavior.

It would also be possible to use a *continuous-time* block, in particular the continuous-time PI controller from Modelica.Blocks.Continuous.PI that is basically implemented as:

```

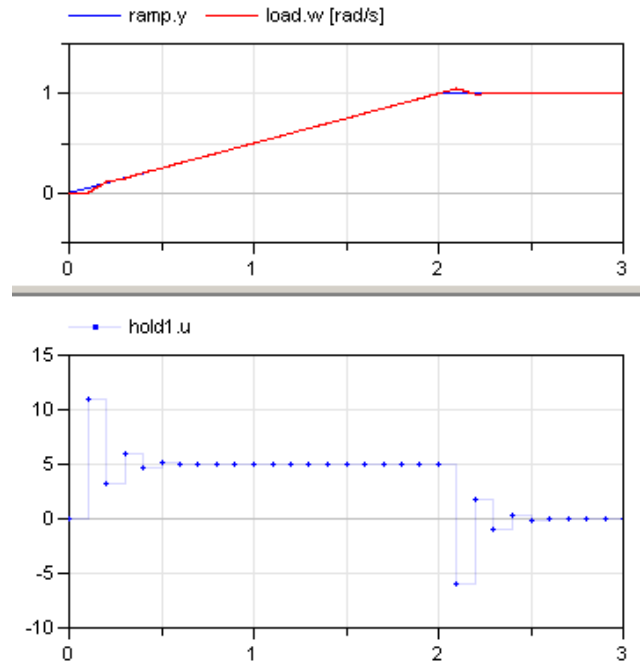
block PI "From Modelica.Blocks.Continuous "
  parameter Real k=1 "Gain";
  parameter Modelica.SIunits.Time T "Time Constant";
  extends Modelica.Blocks.Interfaces.SISO;
  output Real x "State of block";
equation
  der(x) = u/T;
  y = k*(x + u);
end PI;
    
```

In this case the PI controller is described by a differential equation. Since the input signal to this block is a clocked signal when present in the block diagram of Figure 1, the differential equation is automatically discretized by integrating from the previous to the actual clock tick with the integration method defined in component “periodicClock”. In Figure 1, solver “External” is defined (see icon of the clock). This means that the solver defined in the simulation environment is used to integrate the continuous-time block: This might be a variable step-solver with error control where the step size is selected such that it hits the clock tick always exactly.

On the other hand, if solverMethod = ”ImplicitEuler” is selected, then the differential equation of the PI component will be discretized with a fixed step implicit Euler method. This approach is also called “inline integration”. For details, see (Elmqvist *et.al.* 1995). In this case exactly the same result will be obtained as with the previous two PI components.

This approach is very powerful, since *every linear or non-linear continuous-time block* can be utilized in the clocked partition. It is therefore in many cases is no longer necessary to derive discretized blocks manually as, e.g., done in the Modelica_Linear-Systems2.Controller library (Baur *et.al.* 2009).

Typical simulation results are shown in the next figure. Note, here it is clearly visualized by Dymola, that the input to hold1 (= hold1.u) is a clocked signal.

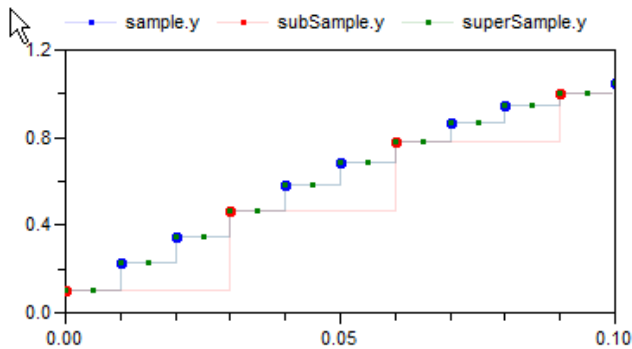


4 SubSample and SuperSample

With blocks “SubSample” and “SuperSample” it can be defined that a partition is sub- or super-sampled with respect to another clocked partition:

	<p>At every “factor” ticks of the input (here: factor = 2), the output ticks and is set to the input.</p>
	<p>At every “factor” ticks of the output (here: factor = 3), the input ticks. The output is set to the last available value of the input.</p>

The factor of a sub- or super-sampled partition can either be explicitly defined with the block, or it can be inferred, since either the factor is defined at another element or exact periods are associated with the partitions (see below). In the next figure an example is shown, where the signal sample.y is sub-sampled by a factor of 3 (= subSample.y) and super-sampled by a factor of 2 (= superSample.y).



There are now many possible ways to define the clocks of time-synchronized partitions. In Figures 2-4 on the next page some useful variants are demonstrated at hand of a cascade control system for a very simple drive system. The goal is that the load inertia travels according to the desired reference angle. This angle is defined with block `KinematicPTP2` from the Modelica Standard Library (the reference signal is constructed so that it moves from a start to an end angle as fast as possible for given maximal speed and maximum acceleration). The “slow” controller part is a simple P-controller to control the position, whereas the “fast” controller part is a PI controller to control the speed.

In Figure 2 one real periodic clock with a sample period of 0.02 s is defined. This clock is then sub-sampled with a factor of 5 which defines a second clock with a sample period of 0.1 s. The “slow” and the “fast” controller partitions are separated by the `super1` block (an instance of `SuperSample`) and therefore it is defined that the output of `super1` is faster than the input of `super1` (the input clock is an integer multiple of the output clock). The two defined clocks are associated with `sample3` and `super1` and therefore the clocks are associated with the partitions “slow controller” and “fast controller”. Note, the factor at `super1` is inferred to be 5.

In Figure 3 only one real clock with a sample period of 0.02 s is defined. This clock is associated to the “fast controller” partition via component `super1`. Now, in component `super1` a factor of “5” is defined. This means that the fast partition is 5-times faster as the slow partition, and therefore the clock of the “slow controller” partition is implicitly defined.

In Figure 4 two “exact” clocks are defined: One clock with a period of 20 ms and one clock with a period of 100 ms. These “absolute” clocks are associated with the “slow” and “fast” partition respectively. Since component `super1` defines that the “fast” partition must be an integer factor faster as the “slow” partition, an implicit constraint is present, that the clocks of the two partitions must have periods that are an integer multiple of each other. Therefore, defining 20 ms and 100 ms is fine. However,

defining periods of 30 ms and 100 ms would result in an error, since this constraint is violated.

The preferred modeling style is a matter of taste. Note, the relative definitions of Figure 2 and Figure 3 have the advantage that parameter `factor` can still be changed after the model is translated (provided a tool supports this feature). Instead, in the definition of Figure 4 it would be typically no longer possible to change the (absolute) periods after translation, since there is a constraint between the two definitions (one period must be an integer multiple of the other period).

5 Nonlinear Inverse Models

Since a long time, Modelica is used to model advanced nonlinear control systems. Especially, Modelica allows a *semi-automatic* treatment of *inverse nonlinear plant models*. In the fundamental article ([Looye et.al. 2005](#)) this approach is described and several controller structures are presented to utilize an inverse plant model in the controller. This approach is attractive because it results in a systematic procedure to design a controller for the whole operating range of a plant. This is in contrast to standard controller design techniques that usually design a linear controller for a plant model that is linearized at a specific operating point. Therefore the operating range of such controllers is inherently limited. Up to Modelica 3.2, controllers with inverse plant models can only be defined as continuous-time systems. Via the export mechanism of Dymola they could be exported with solvers embedded in the code and then used as sampled data system in other environments. However, it is not possible to re-import the sampled data system to Modelica.

The synchronous features of Modelica 3.3 together with the `Modelica_Synchronous` library offer now completely new possibilities, so that the inverse model can be designed and evaluated as sampled data system within Modelica and a Modelica simulation environment such as Dymola. The approach is sketched at hand of a simple nonlinear plant model of a mixing unit ([Föllinger 1998, page 279](#)) and the design of a nonlinear feed-forward controller according to ([Looye et.al. 2005](#)):

A substance A is flowing continuously into a mixing reactor. Due to a catalyst, the substance reacts and splits into several base substances that are continuously removed. The reaction generates energy and therefore the reactor is cooled with a cooling medium. The cooling temperature $T_c(t)$ in [K] is the primary actuation signal. Substance A is described by its concentration $c(t)$ in [mol/l] and its temperature $T(t)$ in [K] according to the following

Simple Drive with cascade controller for position and speed control

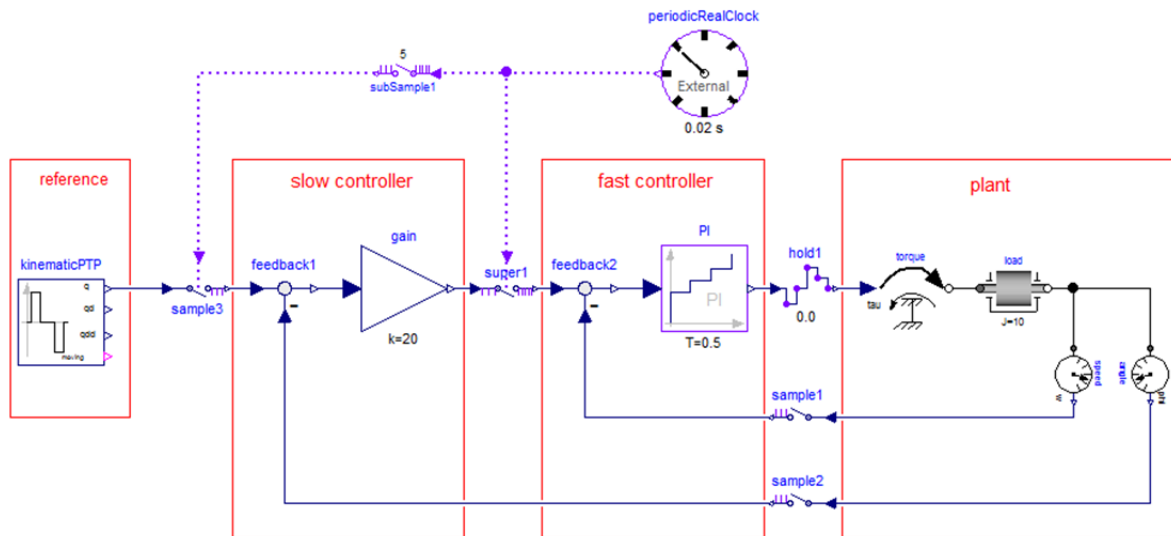


Figure 2: Two clocks are defined with sub-sampling and partitions with super-sampling.

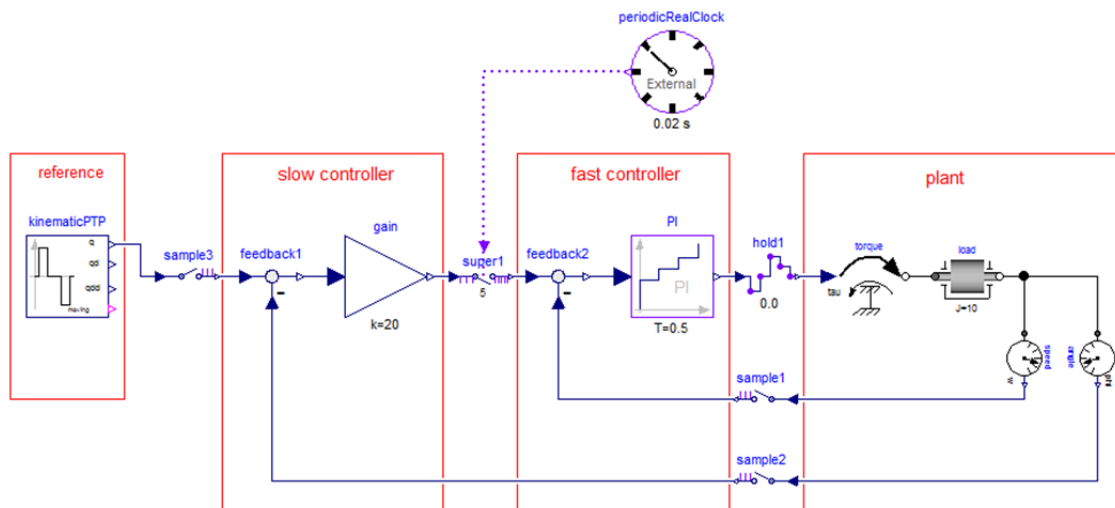


Figure 3: One clock is defined and the second clock is inferred by the factor of the super-sample block.

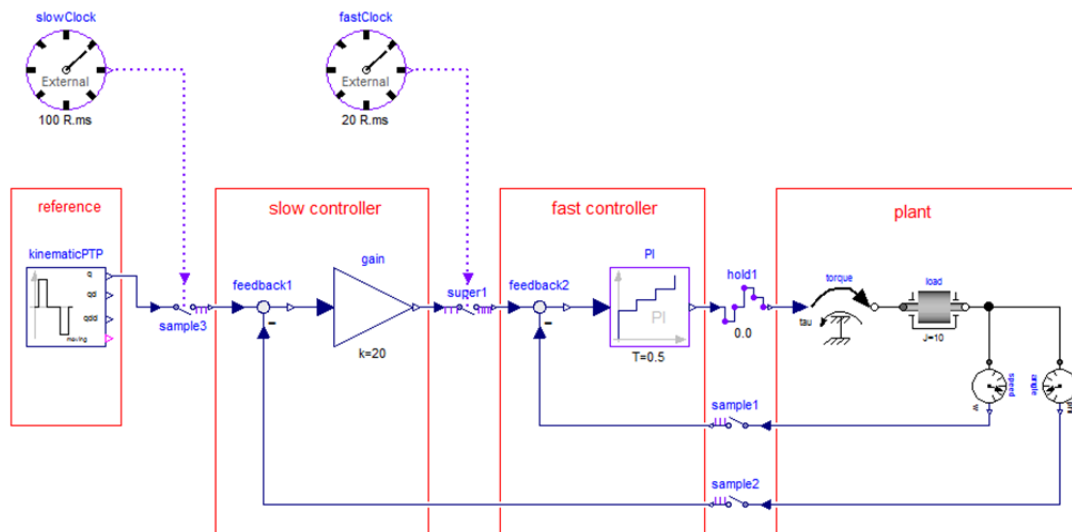


Figure 4: Partitions are defined with exact (integer) clocks that need to be compatible to each other.

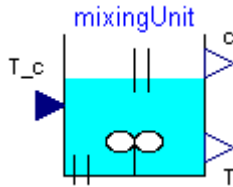
nonlinear differential algebraic equation system:

$$\begin{aligned} \gamma &= c \cdot k_0 \cdot e^{-\varepsilon/T} \\ \dot{c} &= -a_{11} \cdot c - a_{12} \cdot \gamma + a_{13} \\ \dot{T} &= -a_{21} \cdot T + a_{22} \cdot \gamma + a_{23} + b \cdot T_c \end{aligned} \quad (1)$$

with

$$\begin{aligned} k_0 &= 1.24 \cdot 10^{14} & a_{11} &= 0.00446 & a_{21} &= 0.0303 \\ \varepsilon &= 10578 & a_{12} &= 0.0141 & a_{22} &= 2.41 \\ b &= 0.0258 & a_{13} &= 0.00378 & a_{23} &= 1.37 \end{aligned}$$

For the given input $T_c(t)$ these are 1 algebraic equation for the reaction speed $\gamma(t)$ and two differential equations for $c(t)$ and $T(t)$. The concentration $c(t)$ is the signal to be primarily *controlled* and the temperature $T(t)$ is the signal that is *measured*. These equations are collected together in an input/output block:

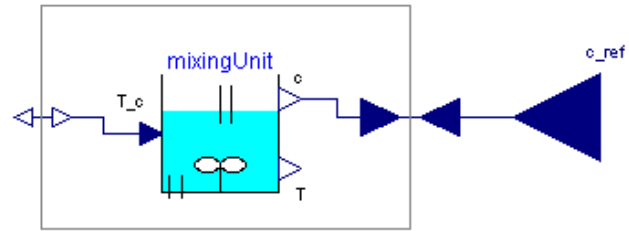


The design of the control system proceeds now in the following steps:

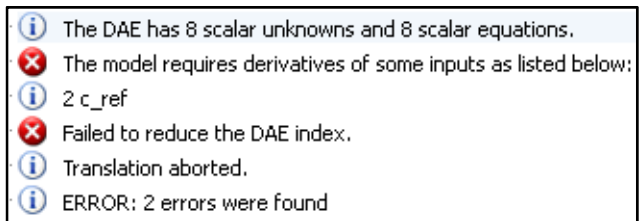
5.1 Design of Pre-Filter

Inverting a model usually means that equations need to be symbolically differentiated and that higher derivatives of the inputs are needed (that are usually not available). One approach is to filter the inputs, so that a Modelica tool can determine the derivatives of the filtered input from the filter states. The minimum needed filter order is determined by first inverting the continuous-time plant model from the variable to be primarily controlled (here: “ c ”) to the actuator input (here: “ T_c ”). This is performed with the help of block “Modelica.Blocks.Math.InverseBlockCons-

traits” that allows connecting an external input (c_{ref} below) to an output (c below):



Translating this model will generate the continuous-time inverse plant model. However, Dymola gives (correctly) an error message:



This message states, that Dymola has to differentiate the model, but this requires the second derivative of the external input c_{ref} and this derivative is not available. The conclusion is that a low pass filter of at least second order has to be connected between c_{ref} and c , for example Modelica.Blocks.Continuous.Filter. Only filter types should be used that do not have “vibrations” in the time domain for a step input. Therefore, parameter analogFilter of the component should be selected as “CriticalDamping” (= only real poles), or “Bessel” (= nearly no vibrations, but steeper frequency response as “CriticalDamping”). The cut-off frequency f_{cut} is manually selected by simulations of the closed loop system. In the example, we use a CriticalDamping filter of third order (the third order is selected to get smoother signals) and a cut-off frequency of 1/300 Hz.

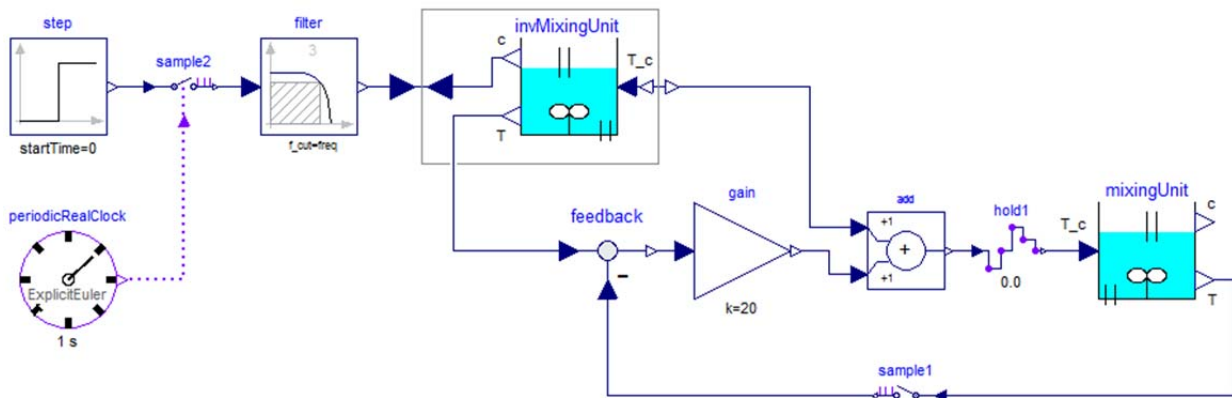


Figure 5: Sampled data controller for mixing unit including the inverse plant model.

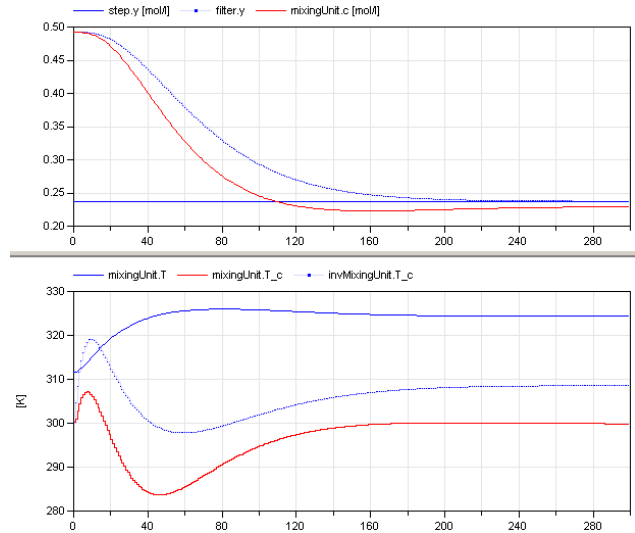
5.2 Design of Controller

The controller for the mixing unit is shown in Figure 5. It consists of the filter discussed in the previous section. The input to the filter is the reference concentration which is filtered by the low pass filter. The output of the filter is used as input to the concentration c in the inverse plant model. This model computes the desired cooling temperature T_c (which is used as desired cooling temperature at the output of the controller) and the desired temperature T (which is used as desired value for the feedback controller). This part of the control system is the “feed forward” part that computes the desired actuator signal. As feedback controller a simple P-Controller with one gain is used.

This controller could be defined as continuous-time system in previous Modelica versions. However, with Modelica 3.3 it is now also possible to define the controller as sampled data system. For this, the two inputs are sampled (`sample1` and `sample2`) and the actuator output is hold (`hold1`).

The controller partition is then associated with a periodic clock (via `sample2`) that has a sample period of 1 s and a solverMethod = “ExplicitEuler”. Since the controller partition is a continuous-time system, it is discretized and solved with an explicit Euler method at every clock tick (by integrating from the previous to the actual time instant of the clock).

The controller works perfectly if the same parameters for the plant and the inverse plant model are used (follows perfectly the filtered reference concentration). Changing all parameters of the inverse plant model by 50 % (with exception of ϵ since the plant is very sensitive to it) still results in a reasonable control behavior as shown by the following simulation results (the desired concentration jumps from 0.492 to 0.237):



The piecewise constant (blue) curve in the upper window is the output of the filter (that is, it is the desired concentration). The red curve in the upper window is the concentration of model mixingUnit, which is the concentration in the plant. Obviously, the concentration follows reasonably well the desired one. By using a more involved feedback controller, the control error could be substantially reduced.

6 Event Clocks –Engine Control

All previous sections concentrated on periodic clocks. However, also non-periodic synchronous sampled data systems can be defined with Modelica 3.3. This is demonstrated at hand of a closed-loop throttle control synchronized to the crankshaft angle of an internal combustion engine. This system has the following properties:

- Engine speed is regulated with a throttle actuator.
- Controller execution is synchronized with the engine crankshaft angle.
- The influence of disturbances, such as a change in load torque, is reduced.

The complete system is shown in Figure 6. Block

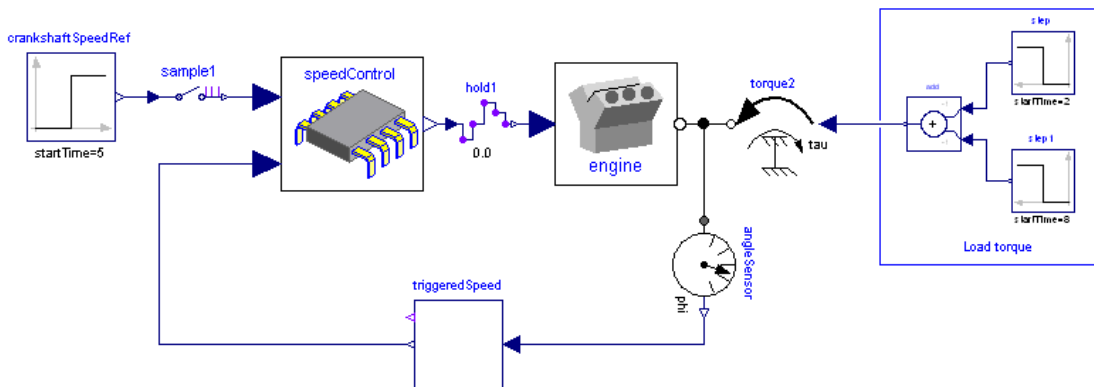


Figure 6: Sampled data engine controller that is synchronized with the crankshaft angle.

`speedControl` is the discrete control system. The boundaries of this controller are defined by `sample1` and `hold1`. A special element `triggeredSpeed` has the crankshaft angle as input and provides the sampled crankshaft speed as output. Additionally, the clock associated with the output (and therefore also to component `speedControl`) ticks, at every 180 degree rotation of the crankshaft angle. This special application is implemented in the text layer of component `triggeredSpeed` as:

```
N = der(angle);
when Clock(angle >= hold(offset)+Modelica.Constants.pi) then
  offset = sample(angle);
  N_clocked = sample(N);
end when;
```

Here, N is the derivative of the crankshaft angle. Whenever this angle becomes larger as 180 degree an event clock is activated due to `Clock(...)`. In such a case the when-clause becomes active, and the speed N is sampled, and the new offset for the next event is computed.

7 Interfaces to External Devices

Bellmann presented in (Bellmann 2009) a Modelica library with capabilities for creating interactive simulation models with advanced (3D-) visualization². It included support for standard input devices such as keyboard and joysticks, as well as communication mechanisms like UDP or shared memory. These device interfaces have been adapted to work with the Modelica synchronous extensions, and have been extended to also support the Linux OS. Furthermore additional functionality such as support for Softing CAN interface cards³ and the (Linux specific) Comedi⁴ control and measurement device interface have been added. In the next figure some of the blocks are shown that are currently available in the external devices library.

8 Cyber-Physical Models

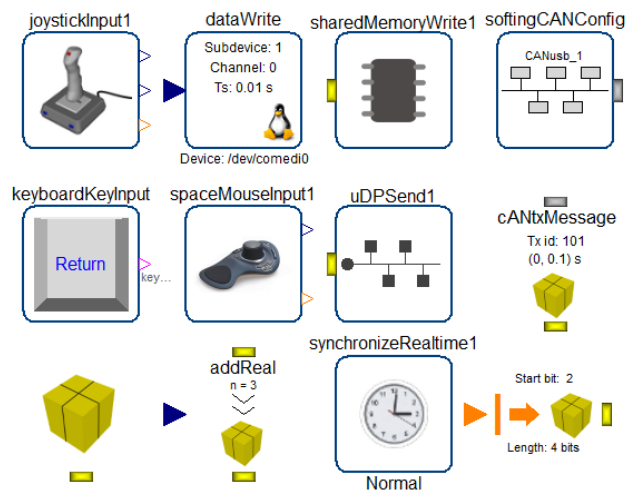
Modelica is designed for modeling of systems containing both physical parts and control systems. It is possible to hierarchically assemble a system out of smart subsystems, i.e. which includes their local control systems.

In (Elmqvist et al. 2009) it is described how parts of the model which is used for evaluating the system

² Today the *visualization* part of that library has evolved into the commercially available product “*Visualization Library*”, which is distributed by BAUSCH-GALL GmbH, <http://www.bausch-gall.de/>.

³ Softing AG (2012), <http://www.softing.com>.

⁴ The Comedi project (2012), <http://www.comedi.org/>.



architecture and performance can be used for different studies and for generation of embedded code. The solution in the Modelica_EmbeddedSystems library is to introduce generic “communication blocks” between the partitions. Such communication blocks can then be configured in different ways, for example to just contain an ideal Sample block or a block with A/D effects. It can also contain a device driver for a A/D converter for the input to the discrete-time partition. It is then possible to use the code of this partition for embedding to control hardware.

If instead, the communication block contains a D/A converter, for the output of the continuous-time partition, the code for the continuous-time partition can be used for hardware-in-the-loop simulation.

The point is that this configuration can be done without changing the original model. It is done by using redeclarations of the content of the communication blocks by using a hierarchical modifier in a model extending the original model. This approach is beneficial with regards of maintaining the original model since only one version is needed.

It is planned that this technique, already evaluated in the Modelica_EmbeddedSystems library, is included in the Modelica_Synchronous library.

9 Summary

A new, free Modelica library is presented that provides a convenient graphical user interface for the synchronous language elements introduced in Modelica 3.3. This library is planned to replace all previous Modelica libraries designed for sampled data systems, since

- the clocking for a partition needs to be defined only at one block (and not at every block of a controller),

- every continuous-time block (including inverse models) can be directly utilized in the clocked partition, thereby making it unnecessary in most cases to provide a manually implemented discrete-time version,
- errors to define the sample periods can be detected by the translator (because all variables and equations of a clocked partition must be associated exactly to one clock),
- more efficient simulation of an overall model consisting of plant (= continuous-time) and controller (= clocked partitions),
- providing the possibility to easily identifying the controller part that shall be downloaded to actual hardware (because all equations and variables of a clocked partition are associated exactly to one clock).

10 Acknowledgement

Sven Erik Mattsson developed the Dymola prototype supporting the synchronous features of Modelica 3.3. Without this prototype, it would not have been possible to develop the Modelica_Synchronous library.

References

- Baur M., Otter M., and Thiele B. (2009): **Modelica Libraries for Linear Control Systems**. Proceedings of 7th International Modelica Conference, Como, Italy, September 20-22.
www.ep.liu.se/ecp/043/068/ecp09430068.pdf
- Benveniste A., Caspi P., Edwards S.A., Halbwachs N., Le Guernic P., and Simone R. (2003): **The Synchronous Languages Twelve Years Later**. Proc. of the IEEE, Vol., 91, No. 1. www.irisa.fr/distribcom/-benveniste/pub/synch_ProcIEEE_2002.pdf
- Bellmann T. (2009): **Interactive Simulations and advanced Visualization with Modelica**. Proceedings of 7th International Modelica Conference, Como, Italy, September 20-22.
www.ep.liu.se/ecp/043/062/ecp09430056.pdf
- Colaco J.-L., and Pouzet M. (2003): **Clocks as First Class Abstract Types**. In Third International Conference on Embedded Software (EMSOFT'03), Philadelphia, Pennsylvania, USA, October 2003.
<http://www.di.ens.fr/~pouzet/lucid-synchrone/papers/emsoft03.ps.gz>
- Dassault Systèmes (2012): Dymola.
<http://www.Dymola.com>
- Elmqvist H., Otter M. and Cellier F.E. (1995): **Inline Integration: A New Mixed Symbolic/Numeric Approach for Solving Differential-Algebraic Equation Systems**. Keynote Address, Proceedings ESM'95, European Simulation Multiconference, Prague, Czech Republic, June 5-8, pp. xxiii-xxxiv.
<http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=6E666F4221CFED902DCA7BDF8DC51AB6?doi=10.1.1.127.3787&rep=rep1&type=pdf>
- Elmqvist H., Otter M., Henriksson D., Thiele B., Mattsson S.E. (2009): **Modelica for Embedded Systems**, Proceedings 7th Modelica Conference, Como, Italy, Sep. 20-22.
<http://www.ep.liu.se/ecp/043/040/ecp09430096.pdf>
- Elmqvist H., Otter M., and Mattsson S.E. (2012): **Fundamentals of Synchronous Control in Modelica**. Proceedings of 9th International Modelica Conference, Munich, Germany, Sep. 3-5.
- Föllinger O. (1998): **Nichtlineare Regelungen I**, Oldenbourg Verlag, 8. Auflage.
- Forget J., F. Boniol, D. Lesens, C. Pagetti (2008): **A Multi-Periodic Synchronous Data-Flow Language**. In 11th IEEE High Assurance Systems Engineering Symposium (HASE'08), Dec. 3-5 2008, Nanjing, China, pp. 251-260.
<http://ieeexplore.ieee.org/xpl/articleDetails.jsp?reload=true&arnumber=4708883&contentType=Conference+Publications>
- Looye G., Thümmel M., Kurze M., Otter M., and Bals J. (2005): **Nonlinear Inverse Models for Control**. Proceedings of 4th International Modelica Conference, ed. G. Schmitz, Hamburg, March 7-8.
https://www.modelica.org/events/Conference2005/online_proceedings/Session3/Session3c3.pdf
- Modelica Association (2012): **Modelica Language Specification Version 3.3**.
<https://www.modelica.org/documents/ModelicaSpec33.pdf>
- Pouzet M. (2006): **Lucid Synchrone, Version 3.0, Tutorial and Reference Manual**.
<http://www.di.ens.fr/~pouzet/lucid-synchrone/>

State Machines in Modelica

Hilding Elmqvist¹ Fabien Gaucher² Sven Erik Mattsson¹ Francois Dupont³

¹Dassault Systèmes AB, Ideon Science Park, SE-223 70 Lund, Sweden

²Dassault Systèmes, 84, Allée Galilée, 38330-Montbonnot-St-Martin, France

³Dassault Systèmes, 120, rue René Descartes, 29280 – Plouzané, France

Hilding.Elmqvist@3ds.com Fabien.Gaucher@3ds.com

SvenErik.Mattsson@3ds.com Francois.Dupont@3ds.com

Abstract

The scope of Modelica has been extended from a language primarily intended for physical systems modeling to modeling of complete systems by allowing the modeling of control systems including state machines.

This paper describes the state machines introduced in Modelica 3.3. Any block without continuous-time equations or algorithms can be a state of a state machine. Transitions between such blocks are modeled by a new kind of connections associated with transition conditions.

The paper gives the details for building state machines and includes several examples. In addition, the complete semantics is described using only 13 Modelica equations.

Keywords: Modelica; State Machines; Control;

1 Introduction

The scope of Modelica has been extended from a language primarily intended for physical systems modeling to modeling of complete systems by allowing the modeling of control systems including state machines and enabling automatic code generation for embedded systems.

This paper presents state machines in Modelica. A companion paper (Elmqvist, *et.al*, 2012) describes the fundamental synchronous language primitives introduced for increased correctness of control systems implementation since many more checks can be done at compile time.

The paper describes language elements to define state machines. Any block without continuous-time equations or algorithms can be a state of a state machine. Transitions between such blocks are represented by a new kind of connections associated with transition conditions.

The paper gives the details for building state machines and includes several examples. In addition, the complete semantics is described using only 13 Modelica equations.

2 States and Transitions

Modelica State Machines will be introduced gradually by means of examples.

Modelica block instances without continuous-time equations or algorithms can potentially be states of a state machine. A cluster of block instances at the same hierarchical level which are coupled by **transition** equations constitutes a state machine. All parts of a state machine must have the same clock. One and only one instance in each state machine must be marked as initial by appearing in an **initialState** equation.

2.1 A Simple State Machine

As a first example, consider the trivial state machine of Figure 1.

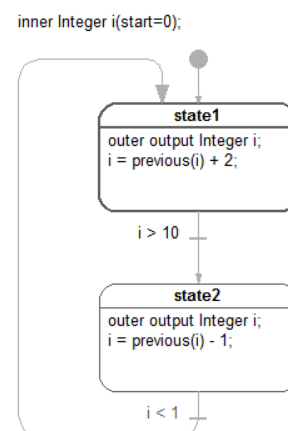


Figure 1. A simple state machine

An inner variable i is defined in the model which has two block instances `state1` and `state2`. In the corresponding block definitions, i is declared as ‘outer output’ which means that i is an output from both of the blocks. In `state1`, i is incremented by 2 and in `state2`, i is decremented by 1. How such multiple definitions are handled is described below.

If `state1` is active, a transition to `state2` is made when $i > 10$. If `state2` is active, a transition to `state1` is made when $i < 1$.

The simulation result is shown in Figure 2.

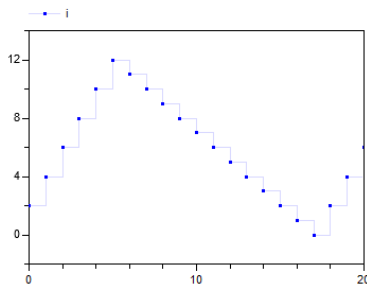


Figure 2. Plot of v of simple state machine

The Modelica code (without annotations) is:

```

model StateMachine1
  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
    initialState(state1);
    transition(state1, state2, i > 10, immediate=false);
    transition(state2, state1, i < 1, immediate=false);
  end StateMachine1;

```

2.2 Merging Variable Definitions

When a state class uses a variable in an **outer output** declaration, the equations have access to the corresponding variable declared **inner**. Special rules are then needed to maintain the single assignment rule since multiple definitions of such outer variables in different mutually exclusive states of one state machine need to be merged.

In each state, the outer output variables (v_j) are solved for (expr_j) and, for each such variable, a single definition is automatically formed:

```

v := if activeState(state1) then expr1
     elseif activeState(state2) then expr2
     elseif ... else last(v)

```

last() is a special internal semantic operator returning its input. It is just used to mark for the sorting that the incidence of its argument should be ignored. A start value must be given to the variable if not assigned in the initial state.

Such a newly created assignment equation might be merged on higher levels in nested state machines.

2.3 Defining a state machine

The following special kinds of connect-equations are used to define transitions between states and to define the initial state:

transition (from, to, condition, immediate, reset, synchronize, priority)
Arguments “from” and “to” are block instances and “condition” is a Boolean expression. The optional arguments “immediate”, “reset”, and “synchronize” are of type Boolean, have parametric variability and a default of true, true, false respectively. The optional argument “priority” is of type Integer, has parametric variability and a default of 1.
This operator defines a transition from instance “from” to instance “to”. The “from” and “to” instances become states of a state machine. The transition fires when condition = true if immediate = true (this is called an “immediate transition”) or previous (condition) when immediate = false (this is called a “delayed transition”).
The argument “priority” defines the priority of firing when several transitions could fire. priority=1 is the highest priority.
If reset = true, the states of the target state are reinitialized, i.e. state machines are restarted in initial state and state variables are reset to their start values.
If synchronize=true, the transition is disabled until all state machines within the from-state have reached the final states, i.e. states without outgoing transitions.
initialState (state)
The argument “state” is the block instance that is defined to be the initial state of a state machine. At the first clock tick of the state machine, this state becomes active.

The attributes of transitions are shown graphically as illustrated in Figure 3.

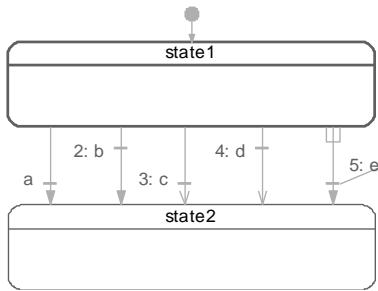


Figure 3. Graphical conventions for transitions

A transition has a perpendicular bar representing the condition which is close to the destination state for an immediate transition, else close to the source state. The arrow is filled for a reset transition otherwise non-filled. A synchronize transition has an “inverted fork” at the source state. Priority is shown preceding the condition if not equal to one. For the 5 transitions in Figure 3, the settings are as follows, from left to right:

- immediate = true, false, true, false, true;
- reset = true, true, false, false, true;
- synchronize = false, false, false, false, true;
- priority = 1, 2, 3, 4, 5.

All transitions leaving the same state must have different priorities.

It is possible to query the status of the state machine by using the following operators:

activeState(state)	Argument “state” is a block instance. The operator returns true, if this instance is a state of a state machine and this state is active at the actual clock tick. If it is not active, the operator returns false. It is an error if the instance is not a state of a state machine.
ticksInState()	Returns the number of clock ticks since a transition was made to the currently active state. This function can only be used in transition conditions of state machines not present in states of higher level state machines.
timeInState()	Returns the time duration as Real in [s] since a transition was made to the currently active state. This function can only be used in transition conditions of state ma-

	chines not present in states of higher level state machines.
--	--

2.4 Immediate and Delayed Transitions

If we attempt to simulate the state machine in Figure 1 with transitions having immediate=true, we get the error message in Dymola:

```
An algebraic loop involving Integers or Booleans has been detected.
```

The reason is that since the transition conditions involve *i*, the variable defined in the equations, there is a cyclic dependency or algebraic loop between the update equations for *i* and the update equations for state machine evolution.

2.5 Conditional Data Flows

An alternative to using outer output variables is to use conditional data flows. Since instances of blocks can be used as states of a state machine, the connection semantics of Modelica has been extended to allow several outputs to be connected to one input.

Consider the combined state machine and data flow diagram in Figure 4:

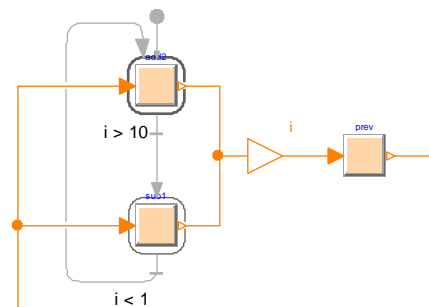


Figure 4. Combined state machine and data flow diagram

The states are instances of the block:

```
block Increment
  extends Modelica.Blocks.Interfaces.PartialIntegerSISO;
  parameter Integer increment;
  equation
    y = u + increment;
  end Increment;
```

with increment values 2 and -1 respectively. The outputs are connected to a protected connector called *i* in order to be able to use *i* in the transition conditions. The connector *i* is connected to an instance of the block:

```
block Prev
  extends Modelica.Blocks.Interfaces.PartialIntegerSISO;
  equation
    y = previous(u);
```

end Prev;

The connections from the state instances to i in Figure 4 are handled in a special way. It is possible to connect several outputs to inputs if all the outputs come from states of the same state machine. In such cases, we get the following constraint equations:

$$u_1 = u_2 = \dots = y_1 = y_2 = \dots$$

with u_i inputs and y_i outputs. The semantics is defined as follows. Introduce a variable v representing the signal flow and rewrite the equation above as a set of equations for u_i and a set of assignment equations for v :

```

v := if activeState(state1) then y1 else last(v);
v := if activeState(state2) then y2 else last(v);
...
u1 = v
u2 = v
...
    
```

The merge of the definitions of v is then made according to section ‘Merging Variable Definitions’. The result of the merge is:

```

v = if activeState(state1) then y1
    elseif activeState(state2) then y2
    elseif ... else last(v)
...
    
```

Plotting i shows the same behavior as the plot of i of the example using inner outer declarations.

3 Hierarchical State Machine Example

Consider the hierarchical state machine in Figure 5:

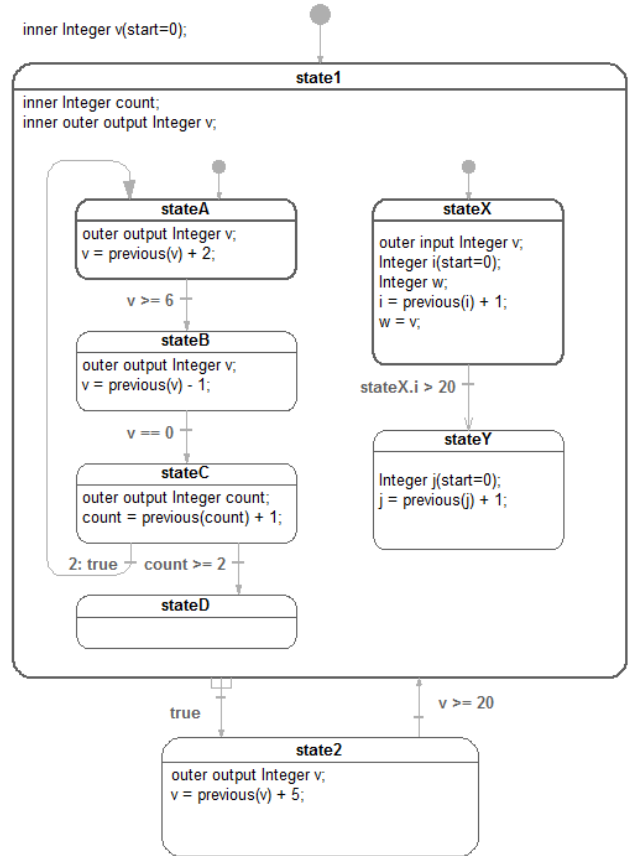


Figure 5. Hierarchical state machine

The model demonstrates the following properties:

- state1 is a meta state with two parallel state machines in it.
- stateA declares v as ‘outer output’. state1 is on an intermediate level and declares v as ‘inner outer output’, i.e. matches lower level outer v by being inner and also matches higher level inner v by being outer. The top level declares v as inner and gives the start value.
- count is defined with a start value in state1. It is reset when a reset transition ($v \geq 20$) is made to state1.
- stateX declares the local variable w to be equal to v declared as ‘inner input’.
- stateY declares a local counter j . It is reset at start and as a consequence of the reset transition ($v \geq 20$) from state2 to state1. However, the reset of j is deferred until stateY is entered by transition ($stateX.i > 20$) although this transition is not a reset transition. This is done by marking that stateY should be reset when making the reset transition $v \geq 20$ and deferring the reset until stateY is actually entered. Synchronizing the exit from the two parallel state machines of state1 is done by using a synchronized transition.

The behavior of the state machine can be seen in the plots of v and w and i of Figure 6:

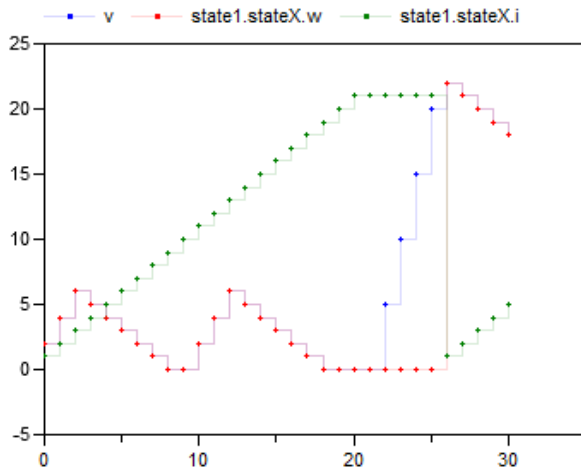


Figure 6. Behavior of hierarchical state machine

4 Adaptive Cruise Control Example

As a more useful example, we will consider a vehicle with adaptive cruise control, i.e. controller that can drive the car at a certain speed or follow the car in front at a safe distance.

The example is simplified considerably to be able to explain all the details in limited space. And the data is just designed for illustrative purposes.

The vehicle dynamics is described by the following model (without annotations):

```

model Vehicle
  parameter Real k=5000;
  parameter Real m=1000;
  parameter Real loss=5;
  Modelica.Blocks.Interfaces.RealInput ud;
  Modelica.Blocks.Interfaces.RealOutput xd;
  Modelica.Blocks.Interfaces.RealOutput vd;

  Modelica.SIunits.Distance x(start=0, fixed=true);
  Modelica.SIunits.Velocity v(start=0, fixed=true);
  Real tau;
equation
  der(x) = v;
  m*der(v) = k*tau - loss*v*abs(v);

  tau = hold(ud);
  xd = sample(x, Clock(1, 10));
  vd = sample(v, Clock(1, 10));
end Vehicle;
    
```

The power train is considered ideal.

A vehicle with the cruise control system is shown in Figure 7. It has an instance of the vehicle dynamics (with a car icon) with a sampled input u_d on the left and two sampled outputs (period=1/10 second), x_d and v_d (counting from the top) to the right.

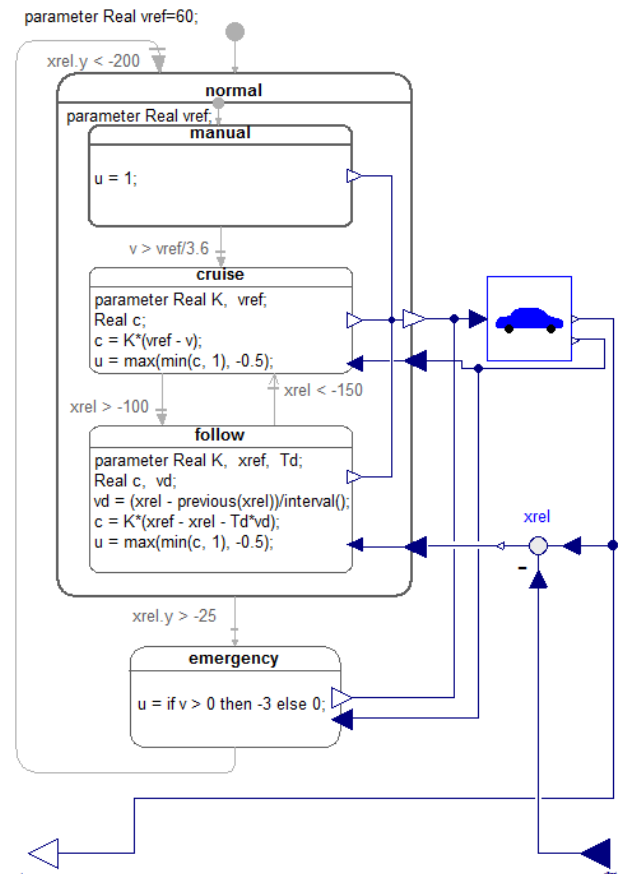


Figure 7. Vehicle with adaptive cruise controller

The top level state machine has two modes: *normal* and *emergency*. Both produces the control signal u connected to u_d of the vehicle. The normal mode has v_d and x_{rel} as inputs. x_{rel} is formed as the difference between the vehicle position and the position of the vehicle in front, available as an input.

The normal state has three states: *manual*, *cruise* and *follow*. The manual state is a simple start up state “stepping on the gas” until the desired speed has been achieved. The cruise state contains a speed controller implemented as a simple P-controller with limitation.

When the vehicle comes within 100 meters of the vehicle in front, *follow* state is entered. It contains a position controller with $x_{ref}=-100$. Since the vehicle is essentially a double integrator from throttle to position, a PD controller is needed. In this case a naïve implementation without filtering is shown. When the distance is larger than 150 meters, *cruise* mode is reentered.

The emergency state is entered when the distance to the car in front is less than 25 meters independently in which substate normal is in. Maximum braking power (-3) is then applied until the car has stopped. When the distance is again 200 meters, the normal

state is entered with a reset transition, i.e. the sub-state of manual of state normal is activated.

The architecture with two entirely different controllers for speed and position was chosen to illustrate the possibility in particular regarding how the data flow connections can be used. (Adaptive cruise control can also be achieved using a cascade controller with an inner speed loop.)

A model of a platoon of 5 CruisingVehicles was built. The desired speed v_{ref} is set as {100, 60, 65, 50, 25} km/h. The initial speeds are the same except for the last car (cruisingVehicle) which is standing still. The distances between the cars are 200 meters.

The results of simulation are shown in Figure 8: position on top and velocity below. All cars slow down to follow the first car (cruisingVehicle4) at 25 km/h at a distance of 100 meter.

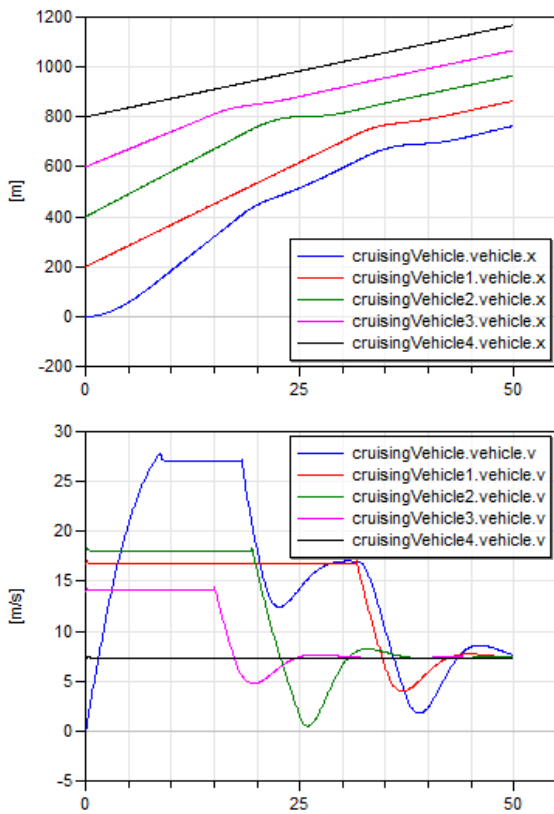


Figure 8. Positions and velocities of vehicles in a platoon

The control signals are shown in Figure 9.

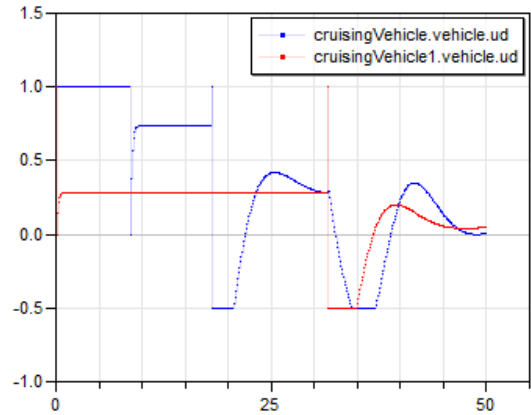


Figure 9: Control signals

The implementation of the cruise state shown in Figure 7 is a bit simplified using a parameter v_{ref} for the velocity set point. Usually, the triggering of going from manual to cruise mode is done by a button. The cruise mode is then picking up the current speed and uses that as a set point. Such an implementation can be made as follows:

```

model Cruise
  parameter Real K = 1;
  Real c, vref;
  Boolean reinit(start=true) = false;
  Modelica.Blocks.Interfaces.RealOutput u;
  Modelica.Blocks.Interfaces.RealInput v;
equation
  vref = if previous(reinit) then v else previous(vref);
  c = K*(vref-v);
  u = max(min(c, 1), -0.5);
end Cruise;

```

This is a general modeling idiom for special treatment when a state is entered. The equation for reinit is $reinit = false$. However, the start value is true, so $previous(reinit)$ gives a pulse at the first cycle if a reset transition is made to the state.

So the desired behavior is achieved by a reset transition from manual to cruise, but a non-reset transition from follow to cruise, since in the last case, the stored v_{ref} should be used.

A platoon of 100 vehicles can easily be constructed using an array of CruisingVehicles:

```

model Platoon
  parameter Integer n=100;
  CruisingVehicle cruisingVehicle[n](vref=linspace(100, 50.5, n));
  Modelica.Blocks.Sources.Constant const(k=10000);
equation
  connect(const.y, cruisingVehicle[n].xFront);
  for i in 1:n-1 loop
    connect(cruisingVehicle[i+1].xd,
            cruisingVehicle[i].xFront);
  end for;
end Platoon;

```

This is a good example of how well the state machine concept is integrated in Modelica allowing to use data flows between states, using modifiers for parameterization, using redeclare of classes and components and using arrays of a mixture of state machines and continuous dynamical models.

5 State Machine Semantics

This section is not intended for normal users of Modelica state machines. It is included since the precise semantics can be described using only 13 Modelica equations and is thus a convenient reference for advanced users and tool developers.

For the purpose of defining the semantics of state machines, assume that the data of all transitions are stored in an array of records, `t`:

```
record Transition
  Integer from;
  Integer to;
  Boolean immediate = true;
  Boolean reset = true;
  Boolean synchronize = false;
  Integer priority = 1;
end Transition;
```

The transitions are sorted with lowest priority number last in the array. The states are enumerated from 1 and up. The transition conditions are stored in a separate array `c[:]` since they are time varying.

The semantics model is a discrete-time system with inputs `{c[:], active, reset}`, outputs `{activeState, activeReset, activeResetStates[:]}` and states `{nextState, nextReset, nextResetStates[:]}`. For a top level state machine, `active` is always true. For sub-state machines, `active` is true only when the parent state is active. For a top level state machine, `reset` is true at the first activation only. For sub-state machine, `reset` is propagated from the state machines higher up.

5.1 State Activation

The state update starts from `nextState`, i.e., what has been determined to be the next state at the previous time. `selectedState` takes into account if a reset of the state machine is to be done.

```
output Integer selectedState =
  if reset then 1 else previous(nextState);
```

The integer `fired` is calculated as the index of the transition to be fired by checking that `selectedState` is the from-state and the condition is true for an immediate transition or `previous(condition)` is true for a delayed

transition. The `max` function returns the index of the transition with highest priority or 0.

```
Integer fired =
  max(if (if t[i].from == selectedState then (if t[i].immediate
then c[i] else previous(c[i])) else false) then i else 0
for i in 1:size(t,1));
```

The start value of `c` is false. This definition would require that the previous value is recorded for all transitions conditions. Below is described an equivalent semantics which just requires to record the value of one integer variable delayed. The integer immediate is calculated as the index of the immediate transition to potentially be fired by checking that `selectedState` is the from-state and the condition is true. The `max` function returns the index of the transition with true condition and highest priority or 0.

```
Integer immediate =
  max(if (if t[i].immediate and t[i].from == selectedState then
c[i] else false) then i else 0 for i in 1:size(t,1));
```

In a similar way, the `Integer delayed` is calculated as the index for a potentially delayed transition, i.e. a transition taking place at the next clock tick. In this case the from-state needs to be equal to `nextState`:

```
Integer delayed =
  max(if (if not t[i].immediate and t[i].from == nextState then
c[i] else false) then i else 0 for i in 1:size(t,1));
```

The transition to be fired is determined as follows, taking into account that a delayed transition might have higher priority than an immediate:

```
Integer fired = max(previous(delayed), immediate);
```

`nextState` is set to the found transitions to-state:

```
Integer nextState = if active then (if fired > 0 then t[fired].to
else selectedState) else previous(nextState);
```

In order to define *synchronize* transitions, each state machine must determine which are the final states, i.e. states without from-transitions and to determine if the state machine is in a final state currently:

```
Boolean finalStates[nStates] =
  {max(if t[j].from == i then 1 else 0 for j in 1:size(t,1)) == 0
for i in 1:nStates};
Boolean stateMachineInFinalState = finalStates[activeState];
```

To enable a *synchronize* transition, all the `stateMachineInFinalState` conditions of all state machines within the meta state must be true.

5.2 Reset Handling

A state can be reset for two reasons:

- The whole state machine has been reset from its context. In this case, all states must be reset, and the initial state becomes active.
- A reset transition has been fired. Then, its target state (and its sub-state machines) are reset, but not other states.

The first reset mechanism is handled by the `activeResetStates` and `nextResetStates` vectors. The state machine reset flag is propagated and maintained to each state individually:

```
output Boolean activeResetStates[nStates] =
  {if reset then true else previous(nextResetStates[i])
  for i in 1:nStates};
```

until a state is eventually executed, then its corresponding reset condition is set to false:

```
Boolean nextResetStates[nStates] = if active then
  {if activeState == i then false else activeResetStates[i]
  for i in 1:nStates}
```

The second reset mechanism is implemented with the `selectedReset` and `nextReset` variables. If no reset transition is fired, the `nextReset` is set to false for the next cycle.

5.3 Activation handling

The execution of a sub-state machine has to be suspended when its enclosing state is not active. This activation flag is given as a Boolean input `active`. When this flag is true, the sub-state machine maintains its previous state, by guarding the equations of the state variables `nextState`, `nextReset` and `nextResetStates`.

5.4 Semantics Summary

The entire semantics model is given below:

```
model StateMachineSemantics "Semantics of state machines"
  parameter Integer nStates;
  parameter Transition t[:];
  "Array of transition data sorted in priority";
  input Boolean c[size(t,1)]
  "Transition conditions sorted in priority";
  Boolean active "true if the state machine is active";
  Boolean reset "true when the state machine should be reset";

  Integer selectedState = if reset then 1 else previous(nextState);
  Boolean selectedReset = if reset then true
    else previous(nextReset);
```

```
// For strong (immediate) and weak (delayed) transitions
Integer immediate = max(if (if t[i].immediate and t[i].from ==
  selectedState then c[i] else false) then i else 0
  for i in 1:size(t,1));

Integer delayed = max(if (if not t[i].immediate and t[i].from ==
  nextState then c[i] else false) then i else 0 for i in 1:size(t,1));

Integer fired = max(previous(delayed), immediate);
output Integer activeState = if reset then 1
  elseif fired > 0 then t[fired].to else selectedState;
output Boolean activeReset = if reset then true
  elseif fired > 0 then t[fired].reset else selectedReset;

// Update states
Integer nextState = if active then activeState
  else previous(nextState);
Boolean nextReset = if active then false
  else previous(nextReset);

// Delayed resetting of individual states
output Boolean activeResetStates[nStates] = {if reset then true
  else previous(nextResetStates[i]) for i in 1:nStates};
Boolean nextResetStates[nStates] = if active then
  {if selectedState == i then false else activeResetStates[i]
  for i in 1:nStates}
  else previous(nextResetStates);

Boolean finalStates[nStates] = {max(if t[j].from == i then 1 else 0
  for j in 1:size(t,1)} == 0 for i in 1:nStates};
Boolean stateMachineInFinalState = finalStates[activeState];
end StateMachineSemantics;
```

6 Comparison to Other State Machine Formalisms

State machines needed to be introduced in Modelica to enable modeling of complete systems. Several attempts have been made: (*Mosterman et al. 1998*), defines state machines in an object-oriented way with Boolean equations. A more powerful state machine formalism was introduced in *StateGraph* (*Otter et al. 2005*). A prototype mode automata formalism was implemented (*Malmheden et al. 2008*) using a built-in concept of modes. Certain problems of potentially unsafe models in *StateGraph* were removed in the *StateGraph2* library (*Otter et al. 2009*). These efforts showed that state machine support must be natively supported in the language.

The presented state machines of Modelica 3.3 have a similar modeling power as *Statecharts* (*Harel, 1987*) and *State Machine Diagrams of SysML* (*Friedenthal 2008*).

The semantics of the state machines defined in this paper is inspired by mode automata (*Maraninchi 2002*) and basically the same as *Lucid Synchrone 3.0* (*Pouzet 2006*), or its clone *LCM* (Logical Control Module) (*Gaucher et al. 2009*). Some minor properties are different compared to *Lucid Synchrone 3.0*,

in particular regarding transition conditions. Lucid Synchrone has two kinds of transitions: namely “strong” and “weak”. Strong transitions are executed before the actions of a state are evaluated while weak transitions are executed after. This can lead to surprising behavior, because the actions of a state are skipped if it is activated by a weak transition and exited by a true strong transition. For this reason, the state machines in Modelica use “immediate” (= the same as “strong”) and “delayed” transitions. Delayed transitions are “immediate” transitions where the condition is automatically delayed with an implicit **previous(...)**.

Note that safety critical control software in aircrafts is often defined with such kind of state machines, such as using the Scade 6 Tool Suite from Esterel Technologies (*Dormoy 2008*) that provides a similar formalism as Lucid Synchrone, with minor differences such as the ability to associate actions to transitions in addition to states. Scade also provides synchronize semantics by means of synchronization transitions between several parallel sub-state machines being in states which have been declared final.

Stateflow (*Mathworks 2012*), while being very expressive, suffers from “numerous, complex and often overlapping features lacking any formal definition”, as reported by (*Hamon, et.al, 2004*).

The presented Modelica approach has the important feature that at one clock tick, there is only one assignment to every variable (for example, it is an error if state machines are executed in parallel and they assign to the same variable at the same clock tick; such errors are detected at compile-time).

Modelica, Lucid Synchrone, LCM and Scade 6 all have the property that data flow and state machines can be mutually hierarchically structured, i.e. that, for example a state of a state machine can contain a block diagram in which the blocks might contain state machines.

7 Conclusions

We have described how state machines can be modeled in Modelica 3.3. Instances of blocks connected by transitions with one such block marked as an initial state constitute a state machine. Hierarchical state machines can be defined with reset or resume semantics, when re-entering a previously executed state. Parallel sub-state machines can be synchronized when they reached their final states. Special merge semantics have been defined for multiple outer output definitions in mutually exclusive states as well as conditional data flows.

8 Acknowledgements

The authors are very thankful to Albert Benveniste, Marc Pouzet, Martin Otter, Martin Malmheden, Daniel Weil, Torsten Blochwitz, Peter Fritzson, Carl-Fredrik Abelson, Hans Olsson and other Modelica Association members for stimulating discussions and feedback during evolutions of the Modelica 3.3 specification.

The authors appreciate the partial funding of this work by the Swedish funding organization VINNOVA (funding number: 2008-02291) within the ITEA2 MODELISAR project (<http://www.itea2.org/project/result/download/result/5533>).

References

- Dormoy F.X. (2008): **SCADE 6 A Model Based Solution For Safety Critical Software Development**, ERTS EMBEDDED REAL TIME SOFTWARE 2008, TOULOUSE, FRANCE, <http://www.esterel-technologies.com/EN-50128/files/ERTS2008-SCADE-6-A-Model-Based-Solution-For-Safety-Critical-Software.pdf>
- Elmqvist H., Otter M., and Mattsson S.E. (2012): **Fundamentals of Synchronous Control in Modelica**. Proceedings of 9th International Modelica Conference, Munich, Germany, September 3-5.
- Friedenthal S., Moore A., and Steiner R. (2008): **A Practical Guide to SysML –The Systems Modeling Language**, Elsevier Inc.
- Gaucher F., Closse E., Weil D. (2009): **The LCM Language Primer**, Dassault Systèmes Internal Report, Grenoble, France, 2009
- Hamon G., and Rushby J. (2004). **An operational semantics for Stateflow**. In Fundamental Approaches to Software Engineering (FASE)'04, volume 2984 of LNCS, pages 229–243, Barcelona, Spain, 2004. Springer. <http://fm.csl.sri.com/~rushby/papers/sttt07.pdf>
- Harel, D. (1987): **Statecharts: A Visual Formalism for Complex Systems**. Science of Computer Programming 8, 231-274. Department of Applied Mathematics, The Weizmann Institute of Science, Rehovot, Israel. www.inf.ed.ac.uk/teaching/courses/seoc1/-2005_2006/resources/statecharts.pdf
- Malmheden M., Elmqvist H., Mattsson S.E., Henriksson D., and Otter M. (2008): **ModeGraph - A Modelica Library for Embedded Control**

Based on Mode-Automata. B. Bachmann (editor), in Proc. of Modelica'2008 conference, Bielefeld, Germany.
www.modelica.org/events/modelica2008/Proceedings/sessions/session3a3.pdf

Maraninchi, F. and Rémond, Y. (2002): **Mode-Automata: a New Domain-Specific Construct for the Development of Safe Critical Systems.**
<http://www.verimag.imag.fr/~maraninx/SCP2002.html>

MathWorks (2012): **R2012a Documentation - Stateflow**
<http://www.mathworks.com/help/toolbox/stateflow/>

Modelica Association (2012): **Modelica Language Specification Version 3.3.**
<https://www.modelica.org/documents/ModelicaSpec33.pdf>.

Mosterman P., M. Otter, and H. Elmqvist. (1998): **Modeling Petri Nets as Local Constraint Equations for Hybrid Systems using Modelica.** Proceedings of SCSC'98, Reno, Nevada, USA, Society for Computer Simulation International, pp. 314–319.

Otter M., K.-E. Årzén, and I. Dressler (2005): **StateGraph – A Modelica Library for Hierarchical State Machines.** Proceedings of the 4th International Modelica Conference, Hamburg, Germany, ed. G. Schmitz, pp. 569-578.
http://www.modelica.org/events/Conference2005/online_proceedings/Session7/Session7b2.pdf

Otter M., Malmheden M., Elmqvist H., S.E. Mattsson, and C. Johnsson (2009): **A New Formalism for Modeling of Reactive and Hybrid Systems.** Proceedings of the 7th International Modelica Conference, Como, Italy, 20-22 September 2009.
<http://www.ep.liu.se/ecp/043/041/ecp09430108.pdf>

Pouzet M. (2006): **Lucid Sychrone, Version 3.0, Tutorial and Reference Manual.**
<http://www.di.ens.fr/~pouzet/lucid-sychrone/>

PNlib - An Advanced Petri Net Library for Hybrid Process Modeling

Sabrina Proß

Bernhard Bachmann

University of Applied Sciences, Department of Engineering and Mathematics

Am Stadtholz 24, 33609 Bielefeld

sabrina.pross@fh-bielefeld.de Bernhard.bachmann@fh-bielefeld.de

Abstract

A new Petri net library, called PNlib, is presented to enable graphical hierarchical modeling, hybrid simulation, and animation of processes in life sciences, technical applications, among others. In order to model these most different processes, a new powerful and universally usable mathematical modeling concept – xHPN (extended Hybrid Petri Net) – has been established. This formalism is used as specification for the PNlib (Petri Net library) realized by the object-oriented modeling language Modelica. The application of the new environment is demonstrated by three selected examples. The first example demonstrates the representation of functional principles by a model of a Senseo coffee machine and the second one is a model of a printing production process. The third example presents the applicability of modeling business processes. All models are provided as application cases in the library.

Keywords: Petri nets; hybrid modeling; xHPN; process modeling

1 Introduction

The Petri net formalism was first introduced by Carl Adam Petri in 1962 for modeling and visualization of concurrency, parallelism, synchronization, resource sharing, and non-determinism [1]. A Petri net is a graph with two different kinds of nodes, called **transitions** and **places**; thereby, places and transitions are connected by arcs. Every place in a Petri net can contain a non-negative integer number of **tokens**. These tokens initiate transitions to fire according to specific conditions. These firings lead to changes of the tokens in the places.

In the recent years, Petri nets with their various extensions are becoming increasingly popular. They have been proven to be a universal graphical modeling concept for representing different systems in nearly all degrees of abstraction. They support the

qualitative modeling approach as well as the quantitative one. Furthermore, the processes can be modeled discretely as well as continuously, refer to [2]. In addition, discrete and continuous processes can also be combined within a Petri net model to so-called **hybrid Petri nets** first introduced by David and Alla [3]. The Petri net formalism with all its extensions is so powerful that nearly all other formalisms are included. Hence, only one formalism is needed regardless of the approach (qualitative vs. quantitative, discrete vs. continuous vs. hybrid, deterministic vs. stochastic) which is appropriate for the respective system. The Petri net formalism is easy to understand for researchers from different disciplines. It is an ideal way for intuitive representing and communicating data and new knowledge of mechanisms and processes. Furthermore, Petri nets allow hierarchical structuring of models and, therefore, offer the possibility of different detailed views for every observer of the model.

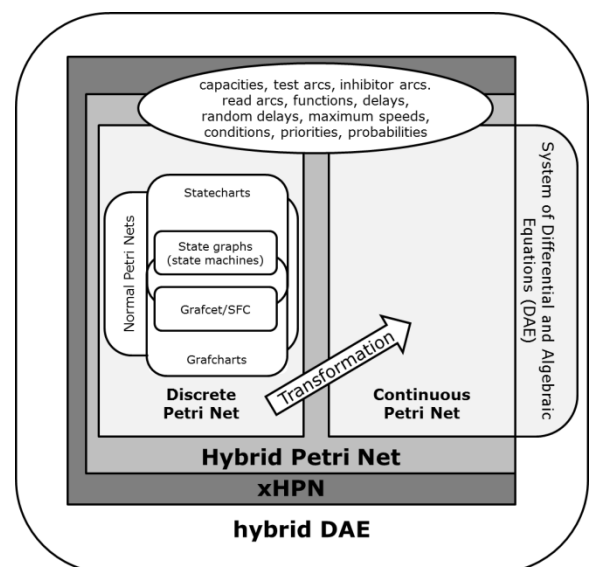


Figure 1: Relationships between the different formalisms

There are already three Petri net libraries available on the Modelica homepage (www.modelica.org). The first was developed by Mosterman et al. and enables the modeling of a restricted class of discrete

Petri nets, called normal Petri nets [4]. The places of normal Petri nets can only contain zero or one token. Additionally, all arcs have the weight one and external signals initiate the firing of transitions. If a conflict occurs between two or more transitions, the transition with the highest priority fires. Hence, only deterministic behavior is represented by this kind of Petri net.

The second Petri net library is an extension of the previous one and was developed by Fabricius [5]. The places are able to contain a non-negative integer number of tokens and can be provided with non-negative integer minimum and maximum capacities. Furthermore, the transitions are timed with fixed or stochastic delays.

The third library, called StateGraph, is based on Grafcharts which combines the function chart formalism of Grafset with the hierarchical states of Statecharts [6]. The StateGraph library is part of the Modelica standard library and was developed by Otter et al. [7].

The relationships between the mentioned concepts are displayed in Figure 1. To enable modeling of different systems with Petri nets in Modelica, the existing libraries have to be extended by the following aspects:

- Transfer of the discrete Petri net concept to a continuous one,
- Support of edges with (functional) weightings,
- Support of test-, inhibitor, and read arcs,
- Support of (different) conflict resolutions (random decisions),
- Combination of discrete and continuous Petri net elements to hybrid Petri nets.

2 Extended Hybrid Petri Nets

The extended Hybrid Petri Net (xHPN) formalism comprises three different processes, called **transitions**: discrete, stochastic, and continuous transition, two different states, called **places**: discrete and continuous places, and four different **arcs**: normal, inhibitor, test, and read arcs. The icons of the formalism are shown in Figure 2.

Discrete places contain a non-negative integer quantity, called **tokens** or **marks**, while continuous places contain a non-negative real quantity. These marks initiate transitions to **fire** according to specific conditions and the firings lead to changes of the marks in the connected places.

Discrete transitions are provided with **delays** and **firing conditions** and fire first when the associated delay is passed and the conditions are fulfilled. The-

se fixed delays can be replaced by exponentially distributed random variables, then, the corresponding transition is called **stochastic transition**. Thereby, the characteristic parameter λ of the exponential distribution can depend functionally on the markings of several places and is recalculated at each point in time when the respective transition becomes active or when one or more markings of involved places change. Based on the characteristic parameter, the next **putative firing time** $\tau = time + Exp(\lambda)$ of the transition can be evaluated and it fires when this point in time is reached.

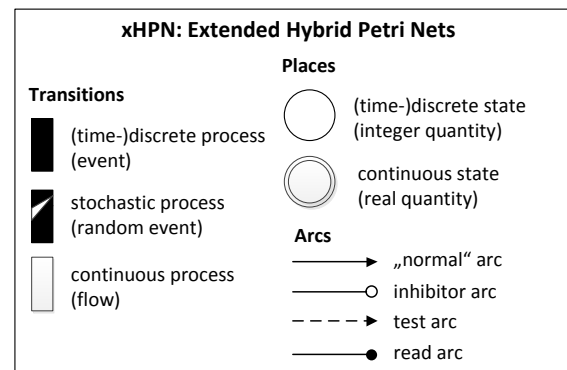


Figure 2: Icons of the xHPN formalism

Both - discrete and stochastic transitions - **fire** by removing the arc weight from all input places and adding the arc weight to all output places. On the contrary, the firing of continuous transitions takes place as a continuous flow determined by the **firing speed** which can depend functionally on markings and/or time.

Places and transitions are connected by **normal arcs** which are weighted by non-negative integers and real numbers, respectively. But also functions can be written at the arcs depending on the current markings of the places and/or time. Places can also be connected to transitions by **test**, **inhibitor**, and **read arcs**. Then their markings do not change during the firing process. In the case of test and inhibitor arcs, the markings are only read to influence the time of firing while read arcs only indicate the usage of the marking in the transition, e.g. for firing conditions or speed functions. If a place is connected to a transition by a test arc, the marking of the place must be greater than the arc weight to enable firing. If a place is connected to a transition by an inhibitor arc, the marking of the place must be less than the arc weight to enable firing. In both cases the markings of the places are not changed by firing.

The **conversion** of a discrete to a continuous marking is realized by connecting a discrete transition to a continuous place and the conversion from a continuous to a discrete marking is realized by con-

necting a continuous place to a discrete transition. However, the conversion process is always performed by discrete transitions, discrete places can only influence the time when continuous transitions fire but their marking cannot be changed during the continuous firing process. Figure 3 shows examples of these two basic principles:

- T1 can only fire when P1 has more than zero marks and P3 has at least one mark (influence),
- T2 can only fire when P4 has at least one mark and P6 has at least 5.4 marks (influence),
- T3 fires by removing one mark from P7 and adding 1.8 marks to P8 (conversion),
- T4 fires by removing 0.8 marks from P9 and adding one mark to P10 (conversion).

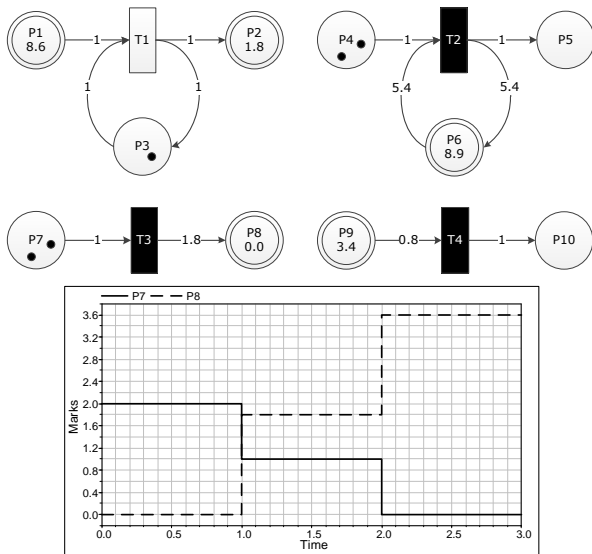


Figure 3: Basic concepts of hybrid Petri nets and marking evolution of places P7 and P8 achieved by firing T3 with a delay of 1 of the bottom left Petri net.

It is important to mention that a discrete transition fires always in a discrete manner by removing and adding marks after a delay is passed regardless of whether a discrete or a continuous place is connected to it. However, a continuous transition fires always by a continuous flow so that a discrete place can only be connected to continuous transition if it is input as well as output of the transition with arcs of same weight. In this way continuous transitions can only be **influenced** by discrete places but discrete markings cannot be changed by continuous firing.

Several conflicts can occur when the places have to enable their connected active transitions. Possibly, a discrete place or a continuous place connected to discrete transitions has not enough marks to enable all discrete output transitions simultaneously or cannot receive marks from all active input transitions due to the maximum capacity. Then a conflict arises that has to be resolved (**type-1-conflict**, see Figure 4).

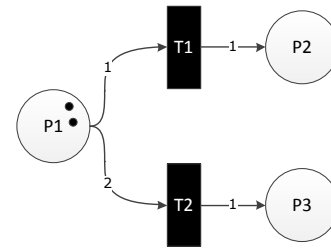


Figure 4: Example of a type-1-conflict; P1 has not enough tokens to fire T1 and T2 simultaneously.

This can be either done by providing the transitions with priorities or probabilities. In the first case, a deterministic process decides which place enables which transition and in the second case the enabling is performed at random; thereby transitions assigned with a high probability are chosen preferentially.

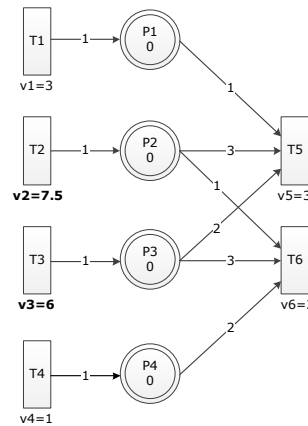


Figure 5: Example of a type-2-conflict; the input speed of P2 and P3 is not sufficient to fire T5 and T6 with the determined speed.

Another conflict can occur between a continuous place and two or more continuous transitions when the input speed is not sufficient to fire all output transitions with the respective speed or when the output speed is not sufficient to fire all input transitions with the respective speed (**type-2-conflict**, see Figure 5). This conflict is solved by sharing the speeds proportional to the assigned maximum speeds (cf. [8]).

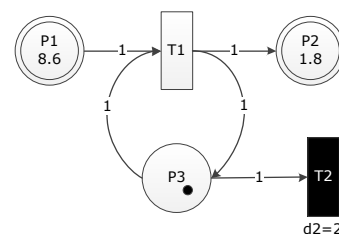


Figure 6: Example of a type-3-conflict; at time 0, T1 becomes active and fires continuously. At time 2, the delay of T2 is passed and it becomes fireable. At this point in time, P3 has a conflict because it cannot fire tokens in T1 and T2, simultaneously. Hence, T2 takes priority over T1 and fires.

If a conflict occurs between a place and continuous as well as discrete/stochastic transitions, the discrete/stochastic transitions take always priority over the continuous transitions (**type-3-conflict**, see Figure 6).

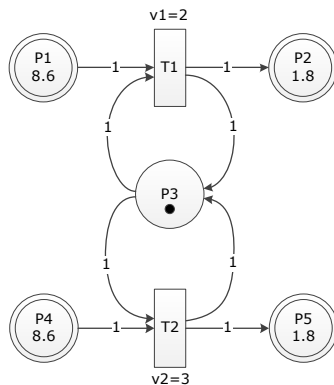


Figure 7: Example of a type-4-conflict; at time 0, P3 can either enable T1 or T2 but not both simultaneously. This conflict can be solved by prioritization of the transitions.

A last conflict can occur when a discrete place has not enough marks to enable all connected continuous transitions. This is solved by prioritization of the involved transitions (**type-4-conflict**, see Figure 7).

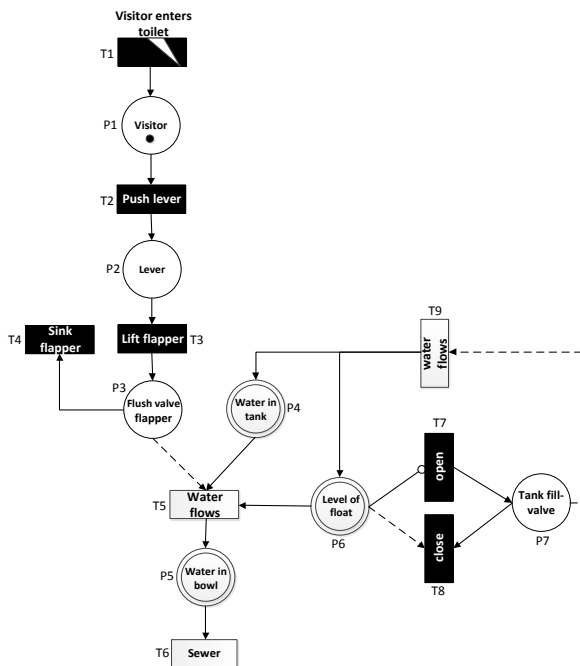


Figure 8: Hybrid modeling of a flush toilet with the aid of xHPN formalism

Figure 8 shows an example of hybrid modeling by the xHPN formalism. The model represents a flush toilet. A visitor enters the toilet; thereby, the time between two visitors is not exactly known so that it is modeled by a stochastic transition with an exponentially distributed delay (T1). The visitor (P1) pushes (T2) the lever (P2) which lifts the flush

valve flapper (P3). Then the water can flow (T5) from the tank (P4) to the bowl (P5) and afterwards to the sewer (T6). When the water flows to the bowl, the float (P6) sinks in the toilet tank. If the float falls below a specific level (inhibitory arc), the tank fill-valve (P7) is opened (T7) and new water can flow (T9) into the tank. This causes also that the float rises and when a specific level is reached (test arc), the tank fill-valve is closed (T8). If the lever has returned to its starting position, the flush valve flapper sinks back to the bottom (T4) and no water can flow into the bowl anymore.

3 PNlib

The advanced Petri Net library, called PNlib, enables the modeling of extended hybrid Petri Nets (xHPN). It comprises

- a discrete (PD) and a continuous place (PC),
- a discrete (TD), a stochastic (TS), and a continuous transitions (TC), and
- a test (TA), an inhibitor (IA), and a read arc (RA).

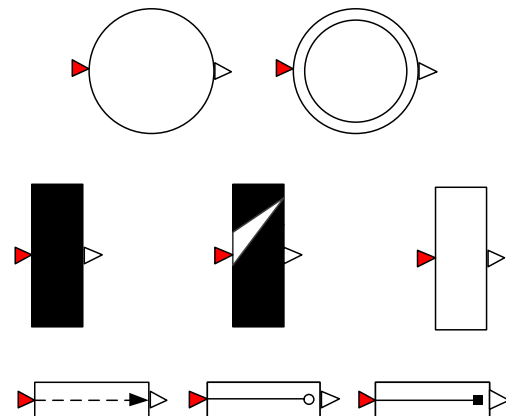


Figure 9: Component icons of the PNlib.

The main package PNlib is divided into the following sub-packages:

- Interfaces: contains the connectors of the Petri net component models.
- Blocks: contains blocks with specific procedures that are used in the Petri net component models.
- Functions: contains functions with specific algorithmic procedures which are used in the Petri net component models.
- Constants: contains constants which are used in the Petri net component models.
- Models: contains several examples and offers the possibility to structure further Petri net models.

Additionally, the package contains the component **settings** which enables the setting of global parameters for the display and the animation of Petri net models.

Places, transitions, and arcs are represented by the icons depicted in Figure 9. Thereby, the discrete place is represented by a circle and the continuous place by a double circle. The transitions are boxes which are black for discrete transitions, black with a white triangle for stochastic transitions, and white for continuous transitions. The test arc is represented by a dashed arc, the inhibitor arc by an arc with a white circle at its end, and the read arc by an arc with a black square at its end.

3.1 Connectors

The PNlib contains four different connectors: `PlaceOut`, `PlaceIn`, `TransitionOut`, and `TransitionIn`. The connectors `PlaceOut` and `PlaceIn` are part of place models and connect them to output and input transitions, respectively. Similar, `TransitionOut` and `TransitionIn` are connectors of the transition model and connect them to output and input places, respectively. Figure 10 shows which connector belongs to which Petri net component model.

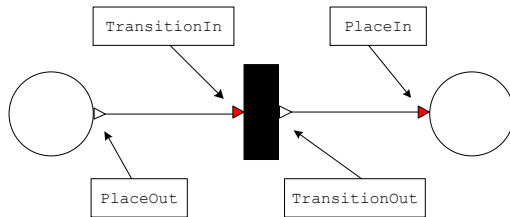


Figure 10: Connectors of the PNlib.

The connectors of the Petri net component models are vectors to enable the connection to an arbitrary number of input and output components. Therefore, the dimension parameters `nIn` and `nOut` are declared in the place and transition models with the `connectorSizing` annotation.

3.2 Places

The parameters of places are summarized in Table 1. If the type-1-conflict is resolved by priorities, the corresponding priorities of the transitions are given by the indices of the connections, i.e. the transition connected to the place with the index 1 has also the priority 1, the transition connected to the place with the index 2 has also the priority 2 etc. Otherwise, if the probabilistic enabling type is chosen, the corresponding probabilities for the transitions have to be entered as a vector. Thereby, the first vector element corresponds to the connection with the index 1, the second to the connection with the index 2 etc. The input of enabling probabilities as vectors in the place model, and not at the corresponding arcs, is necessary due to the fact that properties cannot be as-

signed to connections according to the Modelica Specification 3.2.

Table 1: Parameters and modification possibilities of discrete (d) and continuous (c) places

Name	Type	Default
Description startTokens/ startMarks Marking at the beginning of the simulation	scalar	0
minTokens/ minMarks Minimum capacity	scalar	0
maxTokens/ maxMarks Maximum capacity	scalar	infinite
enablingType Type of enabling if type-1-conflicts occur; the priorities are defined by the connection indices and the probabilities by the variables enablingProbIn/Out	choice/ scalar	Priority
enablingProbIn Enabling probabilities of input transitions	vector	fill(1/nIn,nIn)
enablingProbOut Enabling probabilities of output transitions	vector	fill(1/nOut,nOut)
N Amount of levels for stochastic simulation	scalar	settings1.N
restart Condition for resetting the marking to reStartTokens/Marks	condition expression	false
reStartTokens/ reStartMarks When the reStart condition is fulfilled, the marking is set to reStartTokens/Marks	scalar	0

The input of enabling probabilities as vector is demonstrated by Figure 11. Place P1 is connected to the transitions T1, T2, and T3 and the connection to T1 is indexed by 1, the connection to T2 is indexed by 2, and the connection to T3 is indexed by 3. Thus, the corresponding connect-equations are

```
connect(P1.outTransition[1],
        T1.inPlaces[1]);
connect(P1.outTransition[2],
        T2.inPlaces[1]);
connect(P1.outTransition[3],
        T3.inPlaces[1]);
```

The enabling probabilities 0.3 for T1, 0.25 for T2, and 0.45 for T3 have to be entered by the parameter vector

```
enablingProbOut={0.3,0.25,0.45}.
```

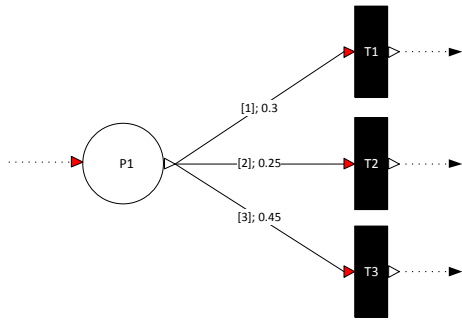


Figure 11: Input of enabling probabilities.

The main process in the place model is the recalculation of the marking after firing a connected transition. In the case of the discrete place model, this is realized by the discrete equation

```

when tokeninout or pre(reStart) then
  t=if tokeninout then pre(t)+
    firingSumIn - firingSumOut else
    reStartTokens;
end when;

```

whereby $pre(t)$ accesses the marking t immediately before the transitions fire. To this amount, the arc weight sum of all firing input transitions is added and the arc weight sum of all firing output transitions is subtracted from it. Additionally, the tokens are reset to $reStartTokens$ when the user-defined condition $reStart$ becomes true.

The marking of continuous places can change continuously as well as discretely. This is implemented by the following construct

```

der(t)=conMarkChange;
when disMarksInOut then
  reinit(t,t+disMarkChange);
end when;
when reStart then
  reinit(t,reStartMarks);
end when;

```

whereby the der -operator access the derivative of the marking t according to time. The continuous mark change is performed by a differential equation while the discrete mark change is performed by the $reinit$ -operator within a discrete equation. This operator causes a re-initialization of the continuous marking every time when a connected discrete transition fires. Additionally, the marking is re-initialized by $reStartMarks$ when the condition $reStart$ becomes true.

3.3 Transitions

The parameters of transitions are summarized in Table 2. Thereby, it has to be distinguished between the following input types: scalar, vector, scalar function, vector function, and condition expression. The input of arc weights as vectors in the transition model and not at the respective arcs is necessary due to the fact

that connections cannot be provided with properties according to the Modelica Specification 3.2.

Table 2: Parameters and modification possibilities of discrete (d), stochastic (s), and continuous (c) transitions

Name Description	Type	Part of	Default Allowed
delay Delay of timed transitions	scalar	d	1 non-negative real values
h Hazard function to determine the characteristic value of exponential distribution	scalar or scalar function	s	1 non-negative real values
maximumSpeed Maximum speed	scalar or scalar function	c	1 non-negative real values
arcWeightIn Weights of input arcs	vector or vector function	d,s,c	1 non-negative integers (d,s), non-negative real values (c)
arcWeightOut Weights of output arcs	vector or vector function	d,s,c	1 non-negative integers (d,s), non-negative real values (c)
firingCon Firing condition	condition expression	d,s,c	true Boolean condition expression

The input is demonstrated by the following examples. Figure 12 shows a discrete Petri net. The indices of the connections are written at the arcs within square brackets, e.g. the connection ($P1 \rightarrow T1$) has the input index [1] and ($T1 \rightarrow P5$) has the output index [3]. The input of the arc weights displayed after the indices to property dialog or as modification equation is performed by the vector functions

$$arcWeightIn = \{2 * P1.t, 4\} \text{ and } arcWeightOut = \{2, 1, 5 * P1.t\},$$

whereby the expression $P1.t$ accesses the current marking of $P1$. Thus, the weights of the arcs ($P1 \rightarrow T1$) and ($T1 \rightarrow P5$) are functions which depend on the marking of $P1$.

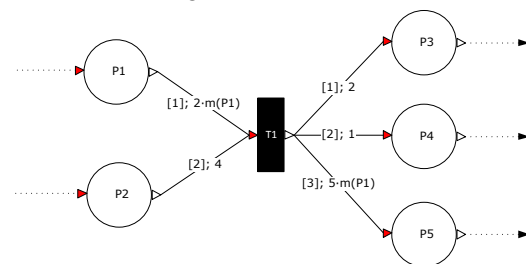


Figure 12: Input of arc weights.

Transitions can also be provided with additional conditions that have to be satisfied to permit the activation. The condition `firingCon = time > 9.7`

causes that the transition cannot be activated as long as time is less than 9.7.

Figure 13 shows two continuous Petri nets. Transition T1 has a maximum speed function which depends on the makings of P1 and P2. The input of this function to the property dialog or as modification equation is performed by the expression

`maximumSpeed = 0.75 * P1.t * P2.t,`

whereby `P1.t` and `P2.t` accesses the marks of P1 and P2, respectively. Transition T2 has a maximum speed function that depends on time and can be entered by the expression

`maximumSpeed = if time <= 6.5 then 2.6 else 1.7.`

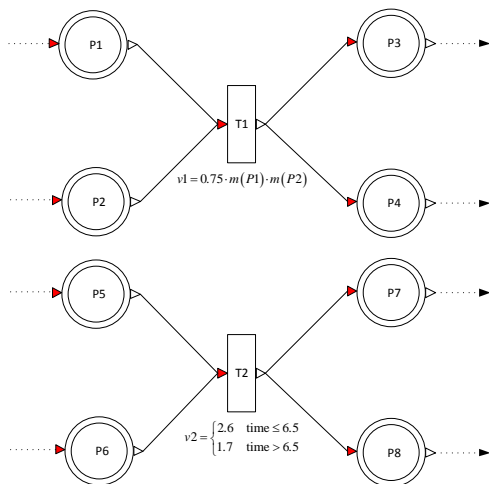


Figure 13: Input of maximum speed functions.

Based on the current markings of the places, it is checked in the transition model by an algorithmic procedure if the transition can become active. Discrete transitions wait then as long as the delay is passed and stochastic transitions wait till the next putative firing time is reached. Based on this information, the places enable some of the active transition to fire. At this point, several conflicts can occur which have to be resolved appropriately by the methods mentioned in [8] to get a successful and reliable simulation. When a transition is enabled by all its connected places, it is firable and reports this via the connector variable `fire` to the connected places. The places recalculate then their markings based on this information.

3.4 Arcs

xHPNs comprise four different kinds of arcs: normal, test, inhibitor, and read arc. The Modelica language do not support the assignment of properties to arcs

that are generated by connect equations. Due to that fact, test, inhibitor, and read arcs are realized by component models which are interposed between places and transitions (see Figure 14); the normal arc is simply generated by the connect equation. Test and inhibitor arc can be normal arcs simultaneously.

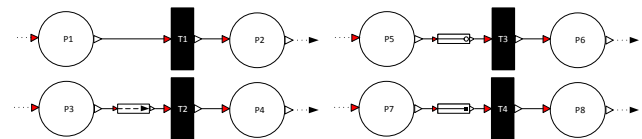


Figure 14: Modeling of normal (top left), test (bottom left), inhibitor (top right), and read arcs (bottom right) with the PNlib.

Table 3: Parameters and modification possibilities of test and inhibitor arcs (read arcs have no parameters)

Name	Type	Default
Description		Allowed
<code>testValue</code> The marking of the place must be greater to enable firing of transitions (test arc); the marking of the place must be smaller to enable firing (inhibitor arc).	scalar	1 non-negative integers if connected to discrete places, non-negative real values otherwise
<code>normalArc</code> If yes is chosen, then the arc is also a normal arc to change the marking by firing (called double arc).	choice/ scalar	no no or yes

4 Animation and Connection to Matlab/Simulink

A possibility to represent the simulation results of an xHPN model is an animation. Thereby, several settings can be made in the property dialog of the settings-box. These settings are global and, thus, affect all components of the Petri net model. By using the prefixes inner and outer, it is achieved that the settings are common to all Petri net components of a model. An animation offers a way to analyze the marking evolutions of large and complex xHPNs. Figure 15 shows four selected points in time of the animation of an xHPN example. All display and animation options are realized with the `DynamicSelect` annotation.

To simulate the established xHPN model several times with different parameter settings and use the arising simulation results for parameter estimation, sensitivity analysis, deterministic and stochastic hybrid simulation, or process optimization [8], the Modelica models in Dymola are connected to

Matlab/Simulink. This is realized with the aid of a Dymola interface in Simulink and a set of Matlab m-files utilities [9].

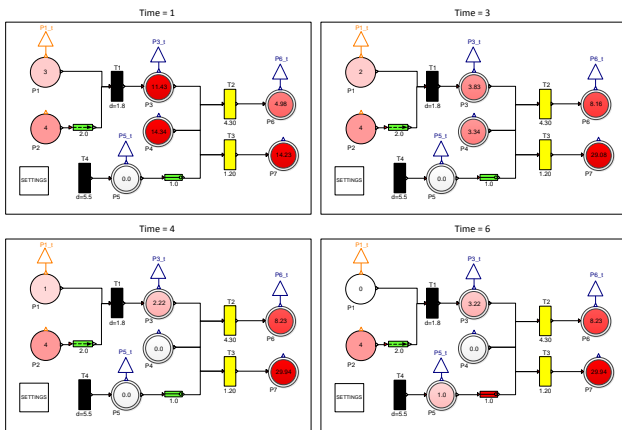


Figure 15: Animation of an xHPN model.

All markings which should be available in Matlab have to be declared with the prefix output on the highest level. This is achieved by creating a connector of the output connector at the top of the place icon. In the case of discrete places it is an orange IntegerOutput connector and in the case of continuous places it is a blue RealOutput connector. In

Figure 15 the markings of P1, P3, P5, and P6 are available in Matlab.

5 Application

The PNlib is so powerful but also so universal and generic that it is an ideal **all-round-tool** for modeling and simulation of nearly all kinds of processes, such as business processes, production processes, logistic processes, work flows, traffic flows, data flows, multi-processor systems, communication protocols, and functional principals. This section gives an overview of the different application fields using the PNlib. Three selected examples

- Modeling a Senseo coffee machine,
 - Modeling a printing process, and
 - Modeling a business process
- are part of the PNlib and should demonstrate the huge application field. Additionally, the application of the PNlib for modeling biological processes is shown in [10].

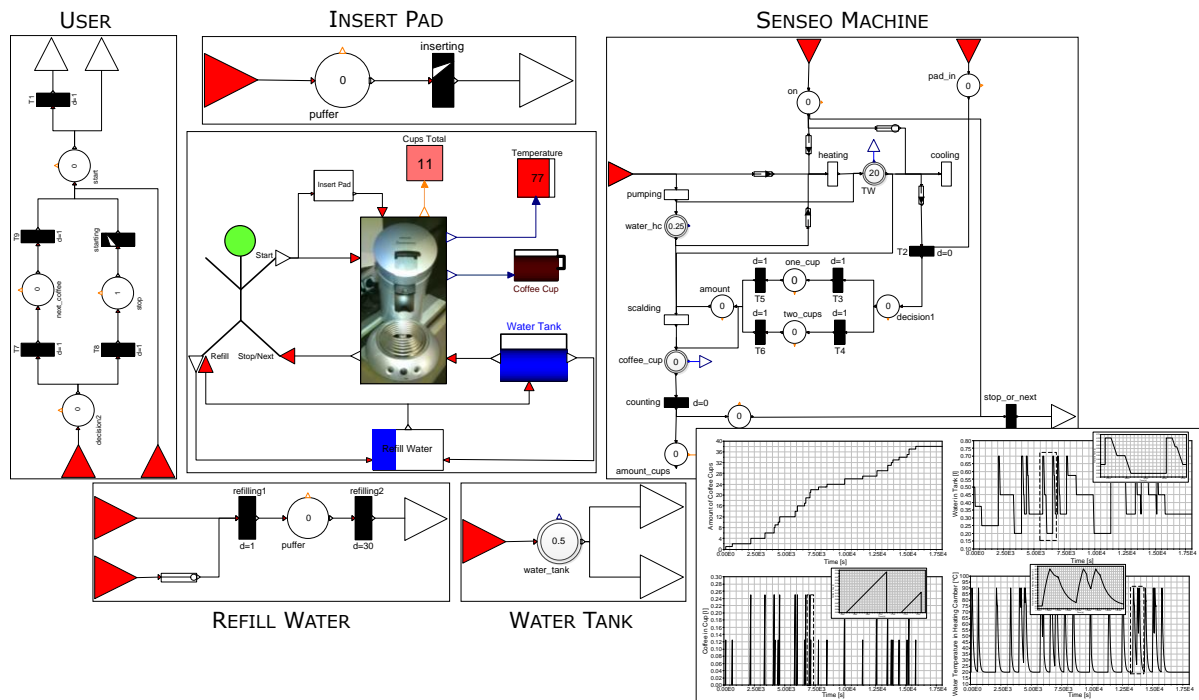


Figure 16: Hierarchical model of a Senseo coffee machine and simulation results.

A model of a Senseo coffee machine is presented. The main feature of a Senseo coffee machine is that the coffee is placed in the machine in a pre-portioned form by so-called coffee pads. One pad is generally used to make one cup of coffee (125°ml) and two pads reach for two cups at 125 ml or one big cup at 250 ml. After a warm-up time of about 60 seconds and the insertion of a coffee pad, the coffee can be made. In this warm-up phase, the water is

heated at 90°C and then pressed with a pressure of about 1.4 bar within 40 seconds through the pad. In contrast to a normal coffee machine that boils the water continuously and transports it by its own buoyancy (hot bubbles) up into the filter, the Senseo machine heats a portion of water completely in a heating chamber and pumps it then through the pad. To ensure that the heating chamber in the machine is always filled with water, a float is placed in the

removable water tank which allows measuring the minimal capacity. If the minimum level is exceeded, the heater is turned off. If there is sufficient water level, the next portion of water is heated directly after the scalding and filling. These functional principles are represented by the hierarchically structured model shown in

Figure 16 and also some simulation results. Additionally, a detailed description of the model can be found in the PNlib.

The applicability of the PNlib for modeling production processes is shown by a model of a printing process. It is also modeled hierarchically to provide a compact and clear view on the highest level containing all important facts (see Figure 17). The process starts with paper on a role and ends with printed leaflets for supermarkets. During the process, misprints, also called maculation, could occur due to several reasons. If the worker at the printing machine detects these misprints, he presses a button and all incorrect exemplars are transferred outward. When the maculation is over, he presses the button again and the process is continued. With the help of this model several new insights can be detected, e.g.

- How and when maculation occurs? What are the causes and how can maculation be prevented?
- How much paper is need for the particular order?
- How long does the order take? ...

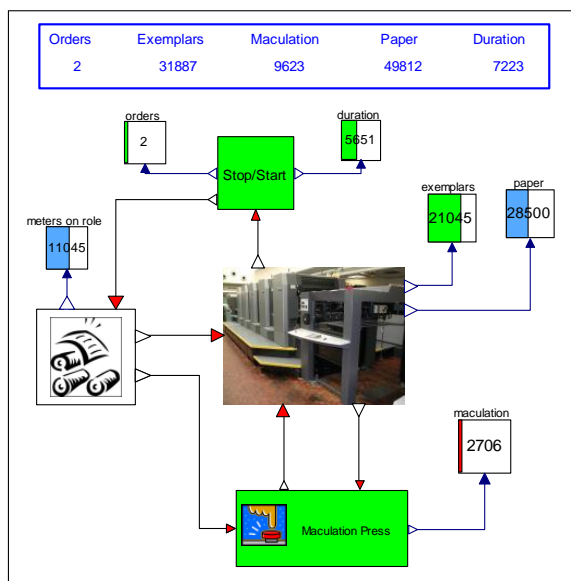


Figure 17: Model of a printing process on the highest level.

The PNlib can also be used for modeling and simulating business processes. A business processes describes a sequence of activities or tasks which have to be carry out in order to achieve a particular business goal e.g. a service or product for a particular customer. Figure 18 shows a small part of a business process model. The major advantages of this approach are (1) the hierarchical structure, which provides

a compact and clear view of the processes on the highest level, and (2) the simulation and animation option which enable analyzing and optimizing of the processes. A possible question may arise in this juncture is, how much employees are needed to accomplish the requests and orders of the customers or simple how the profit can be maximized. All questions of this kind can be answered by simulating the model with different parameter settings.

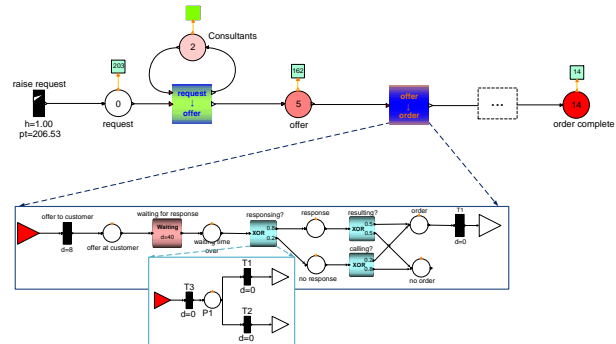


Figure 18: Part of a business process model.

6 Conclusions

A powerful Petri net environment has been developed for graphical hierarchical modeling and hybrid simulation as well as animation of processes from most different application fields. Thereby, the mathematical modeling concept xHPN serves as specification for performing a hybrid simulation. The xHPN elements are modeled object-oriented by discrete, differential, and algebraic equations in the Modelica language. This allows an easy way to maintain, extend, and modify the components.

Moreover, the connection to Matlab/Simulink offers the whole Matlab power for post-processing the simulation results of Modelica models. The Matlab-based tool AMMod (Analysis of Modelica Models) provides already several mathematical methods for data pre-processing, relationship analysis, parameter estimation, sensitivity analysis, deterministic and stochastic hybrid simulation, and process optimization [10].

The application of the new Petri net simulation environment has been demonstrated by a model of a Senseo coffee machine, a model of a printing process, and a model of a business process. All models show the applicability of the xHPN formalism as well as graphical hierarchical modeling and hybrid simulation with the PNlib.

A future goal is to provide an open source Petri-net simulation tool. This demands a further development of the open source Modelica-tool OpenMod-

elica to get the PNlib work with it because some Modelica features are not supported so far.

Moreover, the xHPN formalism as well as the PNlib will be extended by fuzzy logic (e.g. [11]) and the color concept (e.g. [12]) to enhance the range of application fields further.

Furthermore, the PNlib is already connected to VANESA, an open source tool for visualization and analysis of networks, in order to enable modeling, editing, visualization, and animation of xHPN models by an easy-to-use interface [13]. This connection will be further improved.

References

- [1] Petri C.A. Kommunikation mit Automaten. PhD thesis, Rheinisch-Westfälisches Institut für Instrumentelle Mathematik, Bonn, Germany, 1962.
- [2] David R., Alla H. Continuous petri nets. Proceedings of 8th European Workshop on Application and Theory of Petri nets:275-294, 1987.
- [3] David R., Alla H. On Hybrid Petri Nets. Discrete Event Dynamic Systems: Theory and Applications(11): 9–40, 2001.
- [4] Mosterman P.J., Otter M., Elmqvist H. Modeling Petri nets as local constraint equations for hybrid systems using Modelica. Proceedings of SCS Summer Simulation Conference:314–319, 1998.
- [5] Fabricius S.M. Extensions to the Petri Net Library in Modelica. ETH Zurich, Switzerland, 2001
- [6] Johnsson C., Årzén K.-E., Grafchart and grafcet: A comparison between two graphical languages aimed for sequential control applications, Preprints 14th World Congress of IFAC(A): 19-24, 1999.
- [7] Otter M., Årzén K.E., Dressler I. StateGraph-a Modelica library for hierarchical state machines. Proceedings of 4th International Modelica Conference:21-33, 2005
- [8] Proß S. Hybrid Modeling and Optimization of Biological Processes. Bielefeld, Germany, PhD thesis (in preparation), Faculty of Technology, Bielefeld University, Germany, 2012.
- [9] Dynasim AB Dymola-Dynamic Modeling Laboratory-User Manual Volume 2, Lund, Sweden, 2010
- [10] Proß S., Bachmann B. Hybrid Modelling and Process Optimization of Biological Systems, MATHMOD Conference, Wien, Austria 2012.
- [11] Chen S, Ke J, Chang J Knowledge representation using fuzzy Petri nets. Knowledge and Data Engineering, IEEE Transactions on 2(3):311–319, 1990
- [12] Jensen K Coloured petri nets. Petri nets: central models and their properties: 248–299, Springer Verlag, Berlin Heidelberg, 1987
- [13] Proß S., Janowski S. J., Bachmann B., Kaltschmidt C., Kaltschmidt B. PNlib - A Modelica Library for Simulation of Biological Systems based on Extended Hybrid Petri Nets, 3rd International Workshop on Biological Processes & Petri Nets (accepted), Hamburg, Germany, 2012.

Simulation of Non-Newtonian Fluids using Modelica

Pooyan Jahangiri Rita Streblow Dirk Müller
RWTH Aachen University - E.ON Energy Research Center
Mathieustr. 10, 52074 Aachen, Germany
pjahangiri@eonerc.rwth-aachen.de

Abstract

Many fluids used today in different applications show a non-Newtonian behavior. In order to simulate this behavior, many different approaches exist but are not fully implemented in a simulation program. One of the problems with these kinds of simulations is the lack of compatibility with existing models. This makes the modeling very time consuming.

In this paper, a simple approach is shown that provides a general set of equations which can then be used to model both Newtonian as well as non-Newtonian behavior of fluids in the same model in Modelica. Since the implementation is in base models, existing components can easily be used to simulate non Newtonian fluids without sacrificing simulation times.

Keywords: Non-Newtonian; Medium Model; Pressure Drop

1 Introduction

In many applications such as food industries, residential heating and cooling systems, some power plants as well as other energy systems, a non-Newtonian fluid is chosen as the working fluid. The non-Newtonian behavior has a great influence on both flow as well as heat transfer properties of the fluid; therefore, for simulation of such systems, it is necessary to have compatible models and components with this type of fluids.

2 Theory

2.1 The Governing Equations

Non-Newtonian fluids are fluids in which the viscosity changes with respect to the applied stress. According to the correlation between the shear stress and shear rate, fluids can be divided into different categories (see Figure 1).

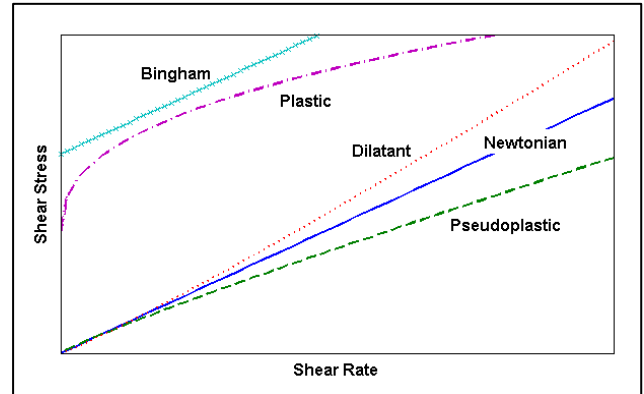


Figure 1: Fluid classification according to shear rate and shear stress

Many available fluids can fully or partly be described by Ostwald-de Waele relationship (Power-law fluids) shown in equation (1).

$$\tau = K\dot{\gamma}^n \quad (1)$$

where

τ	: Shear stress [Pa]
K	: Flow consistency index [Pa.s ⁿ]
$\dot{\gamma}$: Shear rate [s ⁻¹]
n	: Flow behavior index

Flow behavior index “ n ” as well as flow consistency index “ K ” are among the properties of the fluid and are considered constant at a given temperature.

By “ n ” equal 1, the Ostwald-de Waele relationship describes Newtonian fluid behavior. For $n < 1$, Pseudoplastic fluids and for $n > 1$ Dilatant fluids can be described.

2.2 Pressure Drop

For the calculation of the pressure drop in a pipe, when the mass flow rate is known, the dimensionless Darcy friction factor “ λ ” as well as physical parameters of the pipe are used according to equation (2) [1].

$$\Delta p = \lambda \cdot \frac{l}{D} \cdot \frac{\rho \cdot v^2}{2} \quad (2)$$

where

Δp	: Pressure drop [Pa]
λ	: Darcy friction factor
l	: Length of the pipe [m]
D	: Diameter of the pipe [m]
ρ	: Density of the fluid [kg/m ³]
v	: Flow velocity [m/s]

In order to calculate the Darcy friction factor, a Fanning friction factor “ f ” has been introduced by [2] and is shown in equation (3):

$$f = \frac{\lambda}{4} \quad (3)$$

The Fanning friction factor for the laminar region can be calculated from equation (4) and for turbulent region from equations (5) and (6) where “ Re ” corresponds to the Reynolds number. [2, 3, 4]

$$f = \frac{16}{Re} \quad (4)$$

$$f = 0.0014 + \frac{0.125}{Re^{0.32}} \quad (5)$$

$$f = \frac{0.0791}{\sqrt[4]{Re}} \quad (6)$$

2.3 Reynolds Number

In order to calculate the Reynolds number for Power-law fluids, [2] also introduces the general Reynolds number in equation (7).

$$Re = \frac{v^{(2-n)} \cdot D^n \cdot \rho}{\gamma} \quad (7)$$

where

$$\gamma = K' \cdot 8^{n-1} \quad (8)$$

and

$$K' = K \left(\frac{3n+1}{4n} \right)^n \quad (9)$$

Note that using $n=1$, the Reynolds number can be simplified to the Reynolds number in Newtonian fluids as in equation (10).

$$Re = \frac{v \cdot D \cdot \rho}{\mu} \quad (10)$$

Depending on the fluid, the turbulent region can start from Reynolds number between 4000 up to 70,000. On the other hand, the boundary Reynolds number between the laminar and the transitional region can be calculated according to [5] using equation (11).

$$Re_{lam} = \frac{6464n}{(3n+1)^2} \cdot (2+n)^{\frac{2+n}{1+n}} \quad (11)$$

3 Implementation in Modelica

3.1 Existing Flow Models

In Newtonian fluids, the viscosity does not depend on the applied stress or in other words the volume flow rate; hence, it can be calculated within the medium model using just the base properties of the fluid such as the pressure and the temperature.

In existing models in Modelica standard library, at each calculation step, the viscosity of the fluid is calculated within the medium model. This is then used to estimate the Reynolds number according to equation (10). By knowing the Reynolds number, the flow region can be chosen and the governing equations for that region are used to calculate the Darcy friction factor and then the pressure drop.

Note that since Reynolds number is a function of velocity, the procedure mentioned above is only valid when the velocity is known by knowing either the mass flow rate or the volume flow rate and the geometry of the pipe. For this reason, a new set of equations are also implemented to calculate the mass flow rate in a pipe when the pressure difference between two pints is the known variable.

This is helpful for many hydraulic components such as pumps which produce a certain pressure difference and the result will be the flow of the medium; therefore, in order to be able to simulate the flow of the medium properly, it is also necessary to be able to calculate the mass flow rate from the pressure drop.

In this procedure, a second friction factor “ λ_2 ” is introduced which is independent of the velocity and is shown in equation (12).

$$\lambda_2 = \frac{2 \cdot D^3 \cdot \rho}{l \cdot \mu} \cdot \Delta p \quad (12)$$

Using the second friction factor, the Reynolds number can be estimated by the Reynolds number equation for the laminar region and be corrected if the estimation result falls above the turbulent boundary

using the equations governing the turbulent region. The velocity and the mass flow rate are then calculated using the Reynolds equation.

3.2 Non-Newtonian Medium Model

Since the viscosity of Non-Newtonian fluids cannot be calculated using only the base properties, the main calculation should be in the flow model. In order to correlate flow model and medium model with the smallest change possible, an extra function is required in the “*Partial Medium Model*”. This function will describe the flow behavior index “*n*” and is written as follow:

```
replaceable partial function flowBehaviorIndex
  extends Modelica.Icons.Function;
  input ThermodynamicState state "thermodynamic state record";
  output Real n "flow behavior index";
end flowBehaviorIndex;
```

Since the flow behavior index is only a function of the states of the fluid such as the temperature, it can be defined and calculated in the medium model. By adding this partial function for the definition of the flow behavior index in the base medium model, implementing the governing equations or tables for all the fluid models is made possible.

In case the fluid is a Newtonian fluid such as water, the flow behavior index should be set to the constant number of “1”.

By comparing equations (7), (8) and (10) it can be seen that having $n=1$, the coefficient K' is equal the dynamic viscosity “ μ ”. This means that the “dynamicViscosity” function in the partial medium model can also be used to calculate the consistency index K' . Like the flow behavior index, the consistency index does only depend on the base properties of the fluid and not the flow parameters. Therefore it can also be calculated in the medium model.

3.3 Non-Newtonian Flow Models

Having the flow behavior and consistency indices from the medium model, the Reynolds number can be calculated when the volume flow rate is known using the general Reynolds number shown in equation (7). Using the Reynolds number, the fanning friction factor can be calculated using equation (4) for the laminar region and equation (6) for the turbulent region. The pressure drop is then calculated with the help of equations (2) and (3).

For the transitional region between laminar and turbulent, the laminar region is connected to the turbulent region using a cubic Hermite spline.

As already discussed in the existing flow models, it is necessary to have a function for calculation of mass flow with respect to pressure difference in the system. Since pressure drop as well as Reynolds number and hence the Darcy friction factor are a function of velocity which is derived from the mass flow rate, here is also not possible to use the general Reynolds number and Darcy friction factor directly for these calculations. To solve the problem, a variable which is independent of velocity is introduced in equation (13) and is called the modified Darcy friction factor “ λ_m ”.

$$\lambda_m = \lambda \cdot Re^{\left(\frac{2}{2-n}\right)} \quad (13)$$

Combining equations (2), (7) and (13), the modified Darcy friction factor can be calculated as follow:

$$\lambda_m = \frac{2 \cdot D^{\left(\frac{2+n}{2-n}\right)} \cdot \rho^{\left(\frac{n}{2-n}\right)} \cdot \Delta p}{l \cdot \gamma^{\left(\frac{2}{2-n}\right)}} \quad (14)$$

By having a flow behavior index of “1” as for Newtonian fluids, the modified Darcy friction factor is reduced to equation (12).

When the modified Darcy friction factor is known, the Reynolds number can be calculated under the assumption that the flow is laminar using equation (15) achieved from equations (3), (4) and (13).

$$Re = \left(\frac{\lambda_m}{64}\right)^{\left(\frac{2-n}{n}\right)} \quad (15)$$

If the calculated Reynolds number according to equation (15) is greater than the turbulent Reynolds number, then the Reynolds number is calculated for the turbulent region using equation (16) derived from equations (3), (6) and (13).

$$Re = \left(\frac{0.3164}{\lambda_m}\right)^{4\left(\frac{n-2}{n+6}\right)} \quad (16)$$

By knowing the Reynolds number, the mass flow rate can be calculated as follow:

$$\dot{m} = \left(\frac{\left(\frac{\pi}{4}\right)^{2-n} \cdot \gamma \cdot Re}{D^{3n-4} \cdot \rho^{n-1}} \right)^{\frac{1}{2-n}} \quad (17)$$

The transitional region here is also generated by a cubic Hermite spline as before.

4 Simulation Results

The specified functions are directly implemented in the “*detailed wall friction model*”. The model is tested for a Paraffin-Water dispersion shown in Figure 2 with 30% paraffin dispersed in water. The fluid shows a pseudoplastic behavior. The dispersed paraffin goes through a phase change at a certain temperature which not only affects the thermal properties but also the flow properties of the fluid.



Figure 2: Paraffin-Water dispersion used as working fluid in energy systems

The measured flow behavior and consistency indices at different temperatures for the Paraffin-Water dispersion are shown in Figure 3 and are implemented in the medium model.

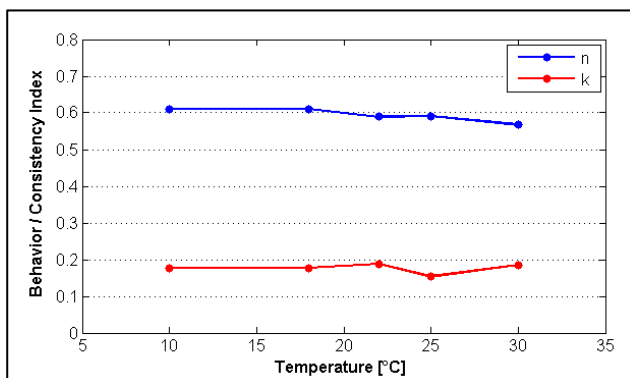


Figure 3: Measured flow behavior and consistency indices for Paraffin-Water dispersion¹

To calculate the pressure drop from a known mass flow rate, Paraffin-Water dispersion model at 22°C, with $n=0.5889$ and $K=0.1877$, is used in a simple simulation model (consisting of a pipe with a length of 1 m and diameter of 0.05 m). The results are shown in Figure 4.

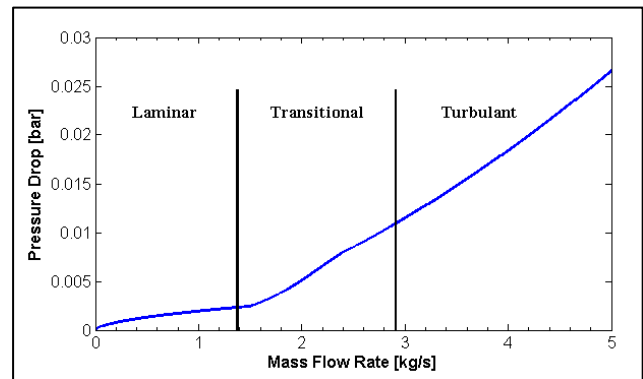


Figure 4: Pressure drop of Paraffin-Water dispersion with respect to mass flow rate

The mass flow rate is then calculated in a simple simulation model when the pressure difference is known for a pipe with 1 m length and 0.05 m diameter and is shown in Figure 5.

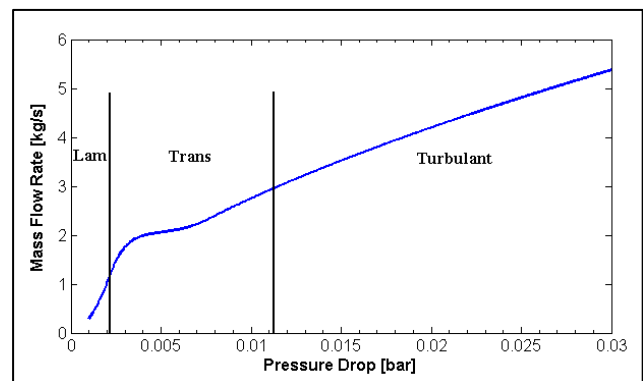


Figure 5: Mass flow rate of Paraffin-Water dispersion with respect to pressure drop

Since the model can be used for all the fluids governed by Ostwald-de Waele relationship, the pressure drop is compared for simulations with 3 different flow behavior indices and is shown in Figure 6. The closer the behavior index to 1 is, the closer the fluid behaves as a Newtonian fluid.

¹ Data provided by Fraunhofer UMSICHT

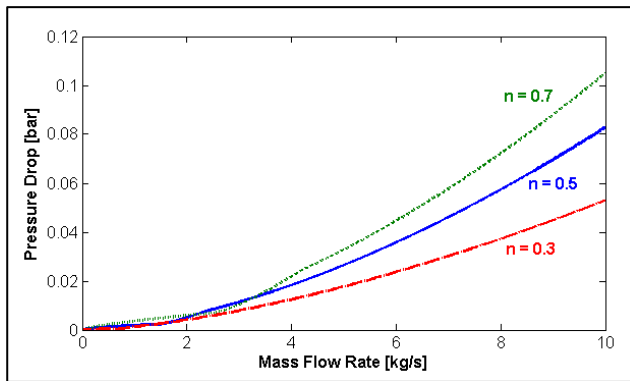


Figure 6: Comparison between different pressure-drops in fluids with different flow behavior indices

4.1 Compatibility

In all the equations described for the non-Newtonian flow, the equations for the Newtonian fluids are derived when the flow behavior index is set to “1”. These are the exact equations which are already implemented in the wall friction model in Modelica standard library. Therefore, the same flow model can be used for all the existing medium models in all existing components without any compatibility issues.

Since the non-Newtonian flow model is an extension to the available flow model in Modelica 3.2, it is also compatible with the entire fluid library. This will omit the need to design new components just for non-Newtonian systems.

4.2 Simulation Times

To substitute the existing model for the general flow model, it is important to maintain the fast simulation speed. Therefore, a simple dynamic simulation is done using Paraffin-Water dispersion and water with the general model described in this paper and is compared to the same simulation setup using water and the existing flow models. The CPU times are compared in Table 1.

Table 1: CPU Time comparison between different models and fluids

CPU Time		
Modified Model		Existing Model
Dispersion	Water	Water
0.106 s	0.109 s	0.105 s

It can be seen that although more complicated equations are used, the solving time stays almost in the same range. This will result in simulation times

which are almost the same as in existing flow models in Modelica.

5 Conclusions

There are many applications for simulating the behavior of non-Newtonian fluids such as food processing plants and energy distribution systems.

In order to implement the non-Newtonian behavior, an extra function is added to the base medium model. This function describes the flow behavior index of a fluid and enables the interaction between the medium model and the flow model. The flow behavior index corresponds to the degree of non-Newtonian behavior of each fluid. Having the necessary interaction between the models, more general equations regarding pressure drops in the system can be implemented. These equations contain both the Newtonian as well as non-Newtonian behavior of a fluid.

Since the changes are in the base models, any other component that uses the model directly or indirectly can be used for the simulation of both Newtonian and non-Newtonian fluids without any additional changes and compatibility issues.

Using the flow behavior index, the non-Newtonian behavior of fluids can later be expanded to the heat transfer properties of fluids in the Modelica thermal libraries.

6 Acknowledgement

Grateful acknowledgement is made for the financial support by BMWi (Federal Ministry of Economics and Technology), promotional reference 032747B and for great help and support by Fraunhofer UMSICHT.

References

- [1] Wagner, W.; Strömung und Druckverlust; 3rd Ed. Germany : Vogel Publication, 1992.
- [2] Metzner, A.B. und Reed, J.C.; Flow of Non-Newtonian Fluids - Correlation of the Laminar, Transition, and Turbulent-flow Regions; A.I.Ch.E. Journal. December 1955, pp. 434-440.
- [3] Dodge, D.W. und Metzner, A.B.; Turbulent Flow of Non-Newtonian Systems; A.I.Ch.E. Journal. June 1959, pp. 189-204.
- [4] Wronski, Jorrit.; Untersuchungen zur Bestimmung des Feststoffgehalts von Dispersionen - Entwicklung eines Ladesensors für PCS-Systeme; Ruhr-Universität Bochum - Lehrstuhl für Feststoffverfahrenstechnik, 2010.
- [5] Chhabra, R.P. und Richardson, J.F; Non-Newtonian Flow in the Process Industries; Oxford, 1999.

HelmholtzMedia — A Fluid Properties Library

Matthis Thorade, Ali Saadat

Helmholtz Centre Potsdam GFZ German Research Centre for Geosciences

Abstract

HelmholtzMedia is a library for the calculation of fluid properties. It is implemented in Modelica and published under the Modelica license. All thermodynamic state properties and their partial derivatives are calculated from a Helmholtz energy equation of state. Further properties that can be calculated include surface tension, viscosity and thermal conductivity.

Keywords: thermodynamic properties, Helmholtz energy, surface tension, viscosity, thermal conductivity

1 Introduction

For the simulation and design of power or refrigeration cycles, accurate properties of the working fluid are indispensable. The most accurate equations of state (EoS) available today for a variety of working fluids are fundamental EoS in terms of Helmholtz energy. From such EoS all thermodynamic state properties, like pressure p or specific entropy s , as well as all partial derivatives of thermodynamic state variables can be calculated.

Further properties of interest are surface tension, viscosity and thermal conductivity. For each of these properties an independent correlation is necessary.

Both the Helmholtz energy EoS as well as correlations for additional properties have been implemented in the HelmholtzMedia library. Details of the implementation are given in the following text.

2 Helmholtz energy fundamental equation of state

A historical overview over the development of fundamental EoS in general is given by [2](in German), an overview over the functional form used today by almost all Helmholtz EoS is given by [8]. The independent variables of the Helmholtz EoS are temperature T and specific volume v or density ρ . Both are non-dimensionalised by their critical values. The

Helmholtz energy f is non-dimensionalised by the specific gas constant R and the temperature T and split up into an ideal gas part α^0 and a residual part α^r . This allows for developing a functional form for the two parts independently.

$$\tau = \frac{T_c}{T}, \quad \delta = \frac{v_c}{v} = \frac{\rho}{\rho_c}, \quad \alpha = \frac{f}{RT} = \alpha^0 + \alpha^r$$

The functional form for the description of the ideal part of the Helmholtz energy results from the thermal equation of state of the ideal gas and a two-fold integration of the heat capacity of the ideal gas. The heat capacity of the ideal gas can be described by polynomial terms, by so-called Planck-Einstein terms or by a combination of the two. Alternatively, hyperbolic functions can be used, but these have not been implemented so far.

$$\begin{aligned} \alpha^0(\delta, \tau) = & \log(\delta) \\ & + \sum_{i=1}^{i=nL} l_{[i,1]} \log[\tau^{l_{[i,2]}}] \\ & + \sum_{i=1}^{i=nP} p_{[i,1]} \cdot \tau^{p_{[i,2]}} \\ & + \sum_{i=1}^{i=nE} e_{[i,1]} \cdot \log[1 - \exp(e_{[i,2]} \cdot \tau)] \end{aligned}$$

The functional form for the description of the residual part of the Helmholtz energy as implemented uses three groups of terms: polynomial terms, so-called Benedict-Webb-Rubin terms and Gaussian bell-shaped terms. For some fluids (e. g. CO₂ or water) the functional form contains additional non-analytical terms

Table 1: Thermodynamic state properties [7]

Property	Algorithm
pressure p	$\rho TR [1 + \delta\alpha_\delta^r]$
entropy s	$R [\tau(\alpha_\tau^0 + \alpha_\tau^r) - (\alpha^0 + \alpha^r)]$
internal energy u	$TR [\tau(\alpha_\tau^0 + \alpha_\tau^r)]$
enthalpy h	$TR [(1 + \delta\alpha_\delta^r) + \tau(\alpha_\tau^0 + \alpha_\tau^r)]$
Gibbs-energy g	$TR [1 + (\alpha^0 + \alpha^r) + \delta\alpha_\delta^r]$

that have not been implemented so far.

$$\alpha^r(\delta, \tau) = \sum_{i=1}^{i=nP} p_{[i,1]} \cdot \delta^{p_{[i,3]}} \cdot \tau^{p_{[i,2]}} + \sum_{i=1}^{i=nB} b_{[i,1]} \cdot \delta^{b_{[i,3]}} \cdot \tau^{b_{[i,2]}} \cdot \exp[-\delta^{b_{[i,4]}}] + \sum_{i=1}^{i=nG} g_{[i,1]} \cdot \delta^{g_{[i,3]}} \cdot \tau^{g_{[i,2]}} \cdot \exp[g_{[i,6]} \cdot (\delta - g_{[i,9]})^2 + g_{[i,7]} \cdot (\tau - g_{[i,8]})^2]$$

A short discussion of all terms is given in [13, Section 5], a very comprehensive discussion is given in [7]. The parameters of the two contributions to the Helmholtz energy are then fitted to experimental data for each fluid. Details on the fitting procedure can be found in [7].

Once the functional form and values for the parameters are known, all state properties can be calculated as simple combinations of the partial derivatives of the Helmholtz energy¹. Algorithms for the calculation of the state properties are given in [7], an extract is repeated in Table 1.

In addition to the state properties, the partial derivatives of state properties are often needed in engineering applications, for example specific heat capaci-

¹The partial derivatives of the Helmholtz energy are abbreviated as follows:

$$\alpha_\tau^0 = \left(\frac{\partial \alpha^0}{\partial \tau} \right)_\delta, \quad \alpha_{\tau\tau}^0 = \left(\frac{\partial^2 \alpha^0}{\partial \tau^2} \right)_\delta, \quad \alpha_{\tau\delta}^0 = \left(\frac{\partial^2 \alpha^0}{\partial \tau \partial \delta} \right)_\delta$$

$$\alpha_\delta^0 = \left(\frac{\partial \alpha^0}{\partial \delta} \right)_\tau, \quad \alpha_{\delta\delta}^0 = \left(\frac{\partial^2 \alpha^0}{\partial \delta^2} \right)_\tau$$

$$\alpha_\tau^r = \left(\frac{\partial \alpha^r}{\partial \tau} \right)_\delta, \quad \alpha_{\tau\tau}^r = \left(\frac{\partial^2 \alpha^r}{\partial \tau^2} \right)_\delta, \quad \alpha_{\tau\delta}^r = \left(\frac{\partial^2 \alpha^r}{\partial \tau \partial \delta} \right)_\delta$$

$$\alpha_\delta^r = \left(\frac{\partial \alpha^r}{\partial \delta} \right)_\tau, \quad \alpha_{\delta\delta}^r = \left(\frac{\partial^2 \alpha^r}{\partial \delta^2} \right)_\tau$$

Table 2: Partial derivatives of state properties [10]

Property	Algorithm
$\left(\frac{\partial p}{\partial \rho} \right)_T$	$TR [1 + 2\delta\alpha_\delta^r + \delta^2\alpha_{\delta\delta}^r]$
$\left(\frac{\partial p}{\partial T} \right)_\rho$	$\rho R [1 + \delta\alpha_\delta^r - \delta\tau\alpha_{\tau\delta}^r]$
$\left(\frac{\partial s}{\partial \rho} \right)_T$	$\frac{R}{\rho} [-(1 + \delta\alpha_\delta^r) + \tau\delta\alpha_{\tau\delta}^r]$
$\left(\frac{\partial s}{\partial T} \right)_\rho$	$\frac{R}{T} [-\tau^2(\alpha_{\tau\tau}^0 + \alpha_{\tau\tau}^r)]$
$\left(\frac{\partial u}{\partial \rho} \right)_T$	$\frac{TR}{\rho} [\tau\delta\alpha_{\tau\delta}^r]$
$\left(\frac{\partial u}{\partial T} \right)_\rho$	$R [-\tau^2(\alpha_{\tau\tau}^0 + \alpha_{\tau\tau}^r)]$
$\left(\frac{\partial h}{\partial \rho} \right)_T$	$\frac{TR}{\rho} [\tau\delta\alpha_{\tau\delta}^r + \delta\alpha_\delta^r + \delta^2\alpha_{\delta\delta}^r]$
$\left(\frac{\partial h}{\partial T} \right)_\rho$	$R [1 - \tau^2(\alpha_{\tau\tau}^0 + \alpha_{\tau\tau}^r) + \delta\alpha_\delta^r - \tau\delta\alpha_{\tau\delta}^r]$
$\left(\frac{\partial g}{\partial \rho} \right)_T$	$\frac{TR}{\rho} [1 + 2\delta\alpha_\delta^r + \delta^2\alpha_{\delta\delta}^r]$
$\left(\frac{\partial g}{\partial T} \right)_\rho$	$R [(\alpha^0 + \alpha^r) + (1 + \delta\alpha_\delta^r) - \tau(\alpha_\tau^0 + \alpha_\tau^r) - \tau\delta\alpha_{\tau\delta}^r]$

ties, the thermal expansion coefficient β , or the isothermal compressibility κ . Any partial derivative can be calculated in a two-step procedure: First, the partial derivatives with respect to *temperature and density*, the independent variables of the EoS, are formed. These are given in [10] and repeated in Table 2. Second, all further derivatives with respect to *arbitrary* state properties can then be transformed into simple combinations of the partial derivatives with respect to temperature and density, using the rules for Jacobian matrix transformations.

For example, the partial derivatives of density with respect to pressure and enthalpy, which are helpful for transient simulation of power cycles, can be expressed as

$$\left(\frac{\partial \rho}{\partial p} \right)_h = \left[\left(\frac{\partial p}{\partial \rho} \right)_T - \left(\frac{\partial p}{\partial T} \right)_\rho \left(\frac{\partial h}{\partial \rho} \right)_T \left(\frac{\partial h}{\partial T} \right)_\rho^{-1} \right]^{-1}$$

and

$$\left(\frac{\partial \rho}{\partial h}\right)_p = \left[\left(\frac{\partial h}{\partial \rho}\right)_T - \left(\frac{\partial h}{\partial T}\right)_\rho \left(\frac{\partial p}{\partial \rho}\right)_T \left(\frac{\partial p}{\partial T}\right)_\rho^{-1} \right]^{-1}.$$

More examples are given in [10].

3 Vapor-liquid equilibrium and two-phase state

The vapour-liquid equilibrium (VLE) of a pure fluid is characterized by three conditions:

$$\begin{aligned} \text{thermal equilibrium: } \Delta T &= (T' - T'') = 0 \\ \text{mechanical equilibrium: } \Delta p &= (p' - p'') = 0 \\ \text{chemical equilibrium: } \Delta g &= (g' - g'') = 0. \end{aligned}$$

For a given temperature T the equilibrium state can be determined by simultaneously solving the equation for mechanical and chemical equilibrium. Using the relations from Table 1 the mechanical equilibrium can be rewritten as

$$\begin{aligned} \Delta p &= \rho' R [1 + \delta' \alpha_\delta^r(\delta', \tau)] \\ &- \rho'' R [1 + \delta'' \alpha_\delta^r(\delta'', \tau)] = 0 \end{aligned}$$

and the chemical equilibrium as

$$\begin{aligned} \Delta g &= TR [1 + \alpha^0(\delta', \tau) + \alpha^r(\delta', \tau) + \delta' \alpha_\delta^r(\delta', \tau)] \\ &- TR [1 + \alpha^0(\delta'', \tau) + \alpha^r(\delta'', \tau) + \delta'' \alpha_\delta^r(\delta'', \tau)] = 0 \end{aligned}$$

resulting in two equations with ρ' and ρ'' as two unknowns. These two equations can be simplified by canceling out the constant and purely temperature-dependent parts and then be solved simultaneously using a Newton-Raphson algorithm as described in [1]. A simplified flowchart for this algorithm is shown in Figure 1. The actual implementation uses dimensionless, scaled variables and gradients.

Once the VLE and the respective saturation states are known, all state properties can be calculated using the vapour mass fraction x . It is defined as

$$x = \frac{m''}{m' + m''} = \frac{\text{mass of vapour}}{\text{mass of liquid} + \text{mass of vapour}}.$$

Using $m = m' + m''$ and $v = V/m$ this can be re-written as

$$x = \frac{v - v'}{v'' - v'} = \frac{1/\rho - 1/\rho'}{1/\rho'' - 1/\rho'}.$$

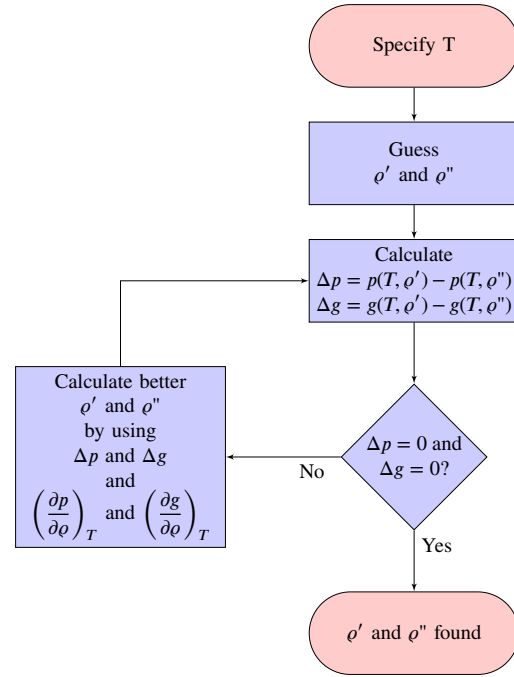


Figure 1: Simplified flowchart for finding the vapour-liquid-equilibrium iteratively, adapted from [1]

Solving for v yields

$$v = xv'' + (1 - x)v' = v' + x(v'' - v').$$

All other state properties can be calculated in the same manner.

In order to calculate the partial derivatives of state properties within the two-phase region, the derivatives along the saturation line are needed. The derivatives of saturation pressure and temperature along the saturation line are given by the Clausius-Clapeyron equation:

$$\begin{aligned} \left(\frac{dp_\sigma}{dT}\right) &= \frac{s'' - s'}{v'' - v'} = \frac{1}{T} \frac{h'' - h'}{v'' - v'} \\ \left(\frac{dT_\sigma}{dp}\right) &= \frac{v'' - v'}{s'' - s'} = T \frac{v'' - v'}{h'' - h'}. \end{aligned}$$

These derivatives can then be used to calculate arbitrary derivatives along the saturation line, and, in a second step, partial derivatives within the two-phase state [10].

4 Iterative procedures

So far, it was assumed that temperature T and density d are known. But the thermodynamic state can as well be defined by specifying any other combination of two independent state variables. In engineering applications,

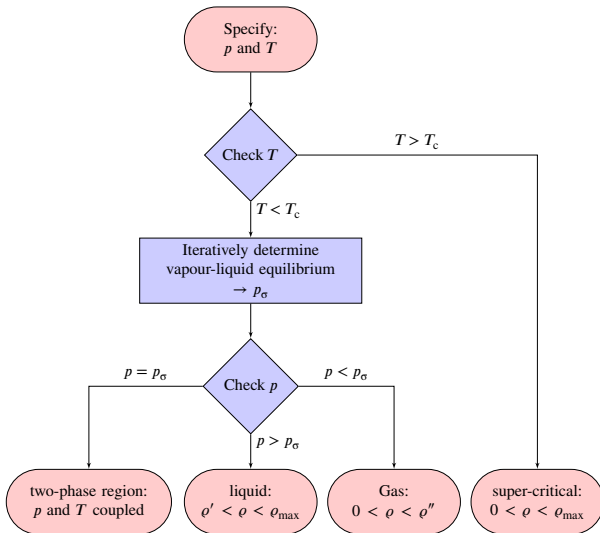


Figure 2: Simplified flowchart for determination of density iteration bounds when pressure and temperature are specified

the known variable combinations often are (p, T) , (p, h) or (p, s) . When any of these combinations is given, the corresponding (T, d) have to be determined iteratively. Two examples of such iterative procedures are given below.

4.1 Density as a function of temperature and pressure

By specifying pressure and temperature, only single-phase states can be described, because in the two-phase region pressure and temperature are not independent. In order to find the density corresponding to the given pressure in the single-phase region, a residual function is defined as

$$RES(\rho) = p - p_{\text{calc}}(\rho, T) \quad .$$

As $p = p(\rho)$ is strictly monotonic for a given temperature, the residual function is as well strictly monotonic and has one single root. Finding the root of the residual function is then equal to finding the density corresponding to the specified pressure. In literature many algorithms for root finding are known, this library uses the algorithm by Brent [3]. It is implemented in the Modelica Standard Library as `Modelica.Math.Nonlinear.solveOneNonlinearEquation`. The mandatory input for this algorithm is a residual function and a lower and upper bound. A flowchart for finding the upper and lower bounds of density is shown in Figure 2.

Once the density is known, all state properties can be calculated using the relations given in Table 1 with

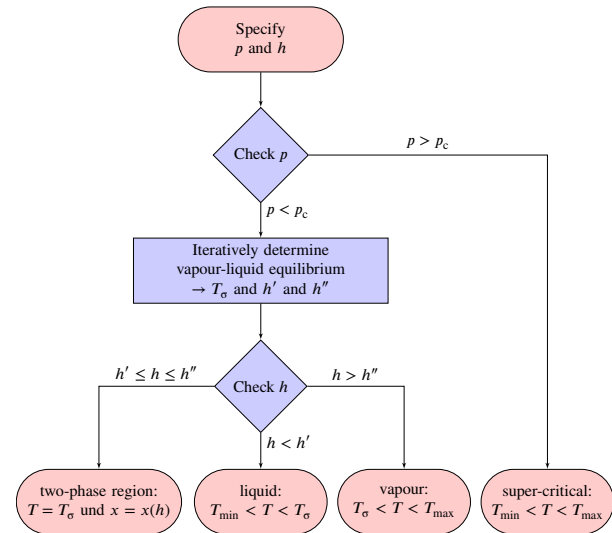


Figure 3: Simplified flowchart for determination of temperature iteration bounds when pressure and enthalpy are specified

density and temperature as input. In the following section, enthalpy and entropy are needed as a function of pressure and temperature. These are calculated by first calculating the density iteratively and then calculating enthalpy and entropy using temperature and density as input variables.

4.2 Density and temperature as a function of pressure and enthalpy

By specifying pressure and specific enthalpy, it is possible to describe single-phase as well as two-phase states. If the pressure is below critical pressure, the first step thus is to determine the vapour-liquid equilibrium corresponding to the specified pressure. The algorithm for VLE determination as described in section 3 uses temperature as input. When the VLE is to be determined from a specified pressure, the residual function

$$RES(T) = p - p_{\sigma, \text{calc}}(T)$$

is used. The lower bound and upper bound for the temperature are the triple temperature and the critical temperature. The VLE information is then used to determine the region and temperature iteration bounds as shown in Figure 3.

Density and temperature can then be determined using the Brent algorithm and the residual function

$$RES(T) = h - h_{\text{calc}}(p, T) \quad ,$$

where $h_{\text{calc}}(p, T)$ already is an iterative function, as described earlier.

5 Ancillary equations

For the determination of the region during the iterative procedures the vapour pressure and the saturation states have to be evaluated. In order to minimize the computational effort, three ancillary equations are given that are sufficiently precise for a first region check. Only if the thermodynamic state is very close to or within the two-phase region the VLE has to be determined from the EoS for best consistency.

Additionally, the results from the ancillary equations are used as start values for the iterative determination of the VLE from the EoS.

5.1 Vapour pressure

The vapour pressure increases sharply with increasing temperature, as shown in Figure 4. The HelmholtzMedia library uses the vapour pressure equation suggested by [12]:

$$\ln \left(\frac{p_\sigma}{p_c} \right) = \frac{T_c}{T} \cdot \sum a_i \left(1 - \frac{T}{T_c} \right)^{n_i} .$$

This vapour pressure equation can be solved for temperature numerically only.

5.2 Density of saturated liquid and saturated vapour

Six models are implemented for the saturated density. These are similar to the models implemented in RefProp [5]. As before the reduced density δ and the reduced inverse temperature τ are defined as

$$\delta = \frac{\rho}{\rho_c} \quad \tau = \frac{T_c}{T} .$$

The reduced density δ at saturation is calculated in a two-step procedure:

$$\Theta = \begin{cases} \left(1 - \frac{T}{T_c} \right) & \text{model 1,3 or 5} \\ \left(1 - \frac{T}{T_c} \right)^{1/3} & \text{model 2,4 or 6} \end{cases}$$

and

$$\delta = \begin{cases} 1 + \sum a_i \Theta^{n_i} & \text{model 1 or 2} \\ \exp \left(\sum a_i \Theta^{n_i} \right) & \text{model 3 or 4} \\ \exp \left(\tau \sum a_i \Theta^{n_i} \right) & \text{model 5 or 6.} \end{cases}$$

Multiplying the reduced density δ with the critical density ρ_c then yields the density ρ .

6 Further properties

6.1 Surface Tension

The surface tension σ between liquid and vapour phase decreases with saturation temperature approaching the critical temperature. This is modeled according to [6]:

$$\sigma = \sum a_i \left(\frac{T_c - T_\sigma}{T_c} \right)^{n_i} .$$

6.2 Viscosity

In this library two viscosity models are implemented that are similar to the models implemented in RefProp [5]. In both models, the viscosity is split into three contributions: the dilute gas viscosity η_0 , the initial density viscosity η_1 and the residual viscosity η_r . This allows for an individual model for each contribution.

$$\eta = \eta_0(T) + \eta_1(\rho, T) + \eta_r(\rho, T) .$$

6.3 Thermal conductivity

One thermal conductivity model has been implemented that is similar to the model implemented in RefProp [5]. The thermal conductivity is split into three contributions: the dilute gas thermal conductivity λ_0 , the residual thermal conductivity λ_r and the critical enhancement contribution λ_c . Each contribution is then individually modeled.

$$\lambda = \lambda_0(T) + \lambda_r(\rho, T) + \lambda_c(\rho, T) .$$

7 Modelica implementation

This library is compatible to and based on Modelica.Media [4]. HelmholtzMedia defines a partial package PartialHelmholtzMedium which extends from Modelica.Media.Interfaces.PartialTwoPhaseMedium. All functions available in the base class are either inherited without modification or they are modified by implementing a new algorithm.

The Record ThermodynamicState contains density, temperature, pressure, specific enthalpy, specific internal energy and specific entropy. Compared to the base class, specific entropy was added. The Record SaturationProperties was modified by adding the states liq and vap.

Where possible, annotation(inverse=...); and annotation(derivative=...); were used.

For fluids that can be modeled by the implemented algorithms, adding a new fluid is done by extending

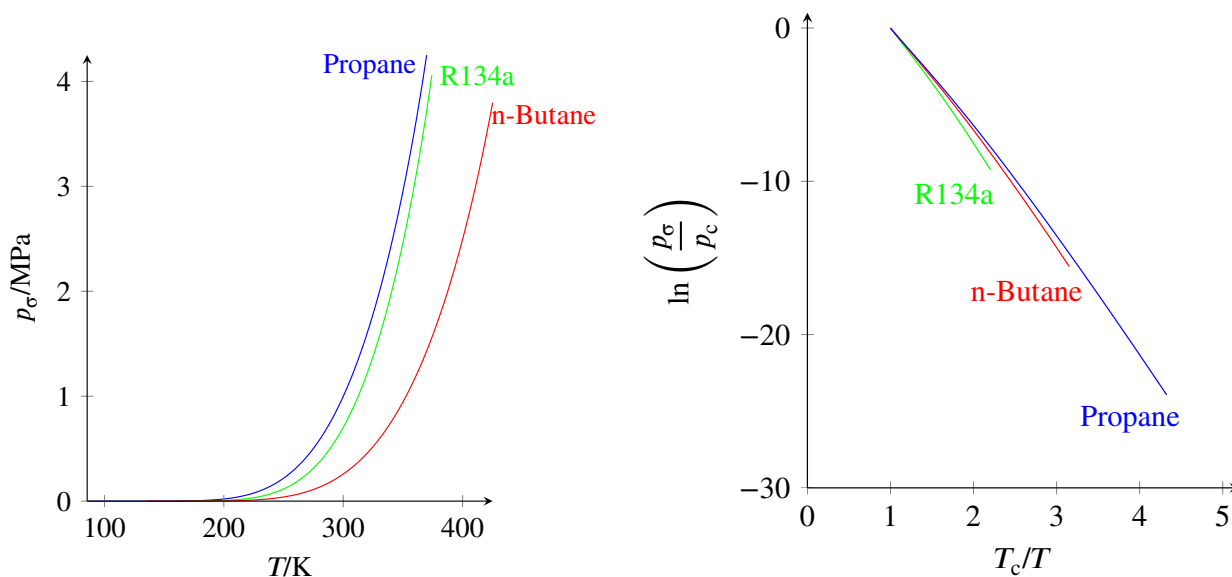


Figure 4: Vapour pressure as a function of temperature

from `PartialHelmholtzMedium` and modifying the parameters for the algorithms. The parameters need to be copied from the respective publications and saved in the format used by `HelmholtzMedia`. `RefProp` [5] comes with a comprehensive compilation of these parameters, so that `RefProp` licencees could alternatively copy them from the `RefProp` fluid files. So far, six fluids have been implemented: n-Butane, Isobutane, Isopentane Propane, R134a and Ethanol. The parameters for these six fluids have been copied from `RefProp`.

8 Summary and Outlook

The most accurate equations of state (EoS) available today for a variety of working fluids are fundamental EoS in terms of Helmholtz energy. The `HelmholtzMedia` library implements the Helmholtz energy EoS in a generalized form that makes adding more fluids very easy. In addition to the equation of state, algorithms for the calculation of viscosity, thermal conductivity and surface tension are given, as well as ancillary equations for saturation properties that speed up iterative procedures. Apart from these ancillary equations, the library is not optimized for speed.

Possible extensions for future versions include the addition of non-analytic terms for the residual part of the Helmholtz energy and hyperbolic terms for the ideal part of the Helmholtz energy. For viscosity and thermal conductivity two more models could be added, an extended corresponding states model and a model based on the generalized friction theory.

In order to add accurate EoS for mixtures like the

GERG-2008 model, a template for multi-component multi-phase media would be necessary. The structure of `Modelica.Media` might change in a future version of the `Modelica Standard Library` [11].

The library is completely written in `Modelica` and released as open-source under the terms of the `Modelica` license. Anybody interested in the library is invited to contribute; the source code and an issue tracker are available at [9].

Acknowledgment

The authors would like to thank Eric W. Lemmon for answering an abundance of questions and for providing a modified version of `RefProp` capable of outputting additional intermediate results.

This work was performed in the framework of the `GeoEn` project and was funded by the Federal Ministry of Education and Research of Germany (Grant 03G0767A).

References

- [1] R. Akasaka. “A Reliable and Useful Method to Determine the Saturation State from Helmholtz Energy Equations of State”. In: *Journal of Thermal Science and Technology* 3.3 (2008), pp. 442–451. DOI: 10.1299/jtst.3.442.

- [2] H. D. Baehr. “Thermodynamische Fundamentalgleichungen und charakteristische Funktionen”. In: *Forschung im Ingenieurwesen* 64.1 (1998), pp. 35–43. DOI: 10.1007/PL00010764.
- [3] R. Brent. *Algorithms for minimization without derivatives*. Prentice-Hall, 1973.
- [4] H. Elmqvist, H. Tummescheit, and M. Otter. “Object-oriented modeling of thermo-fluid systems”. In: *Proceedings of the 3rd International Modelica Conference*. 2003, pp. 269–286.
- [5] E. W. Lemmon, M. L. Huber, and M. O. McLinden. *NIST Standard Reference Database 23: Reference Fluid Thermodynamic and Transport Properties - REFPROP*. 9.0. National Institute of Standards and Technology, Standard Reference Data Program. Gaithersburg, 2010.
- [6] G. R. Somayajulu. “A generalized equation for surface tension from the triple point to the critical point”. In: *International Journal of Thermophysics* 9.4 (1988), pp. 559–566. DOI: 10.1007/BF00503154.
- [7] R. Span. *Multiparameter equations of state: an accurate source of thermodynamic property data*. Springer Verlag, 2000.
- [8] R. Span, W. Wagner, E. W. Lemmon, and R. T. Jacobsen. “Multiparameter equations of state — recent trends and future challenges”. In: *Fluid Phase Equilibria* 183-184.1-2 (2001), pp. 1–20. DOI: 10.1016/S0378-3812(01)00416-2.
- [9] M. Thorade. *HelmholtzMedia*. 2012. URL: <https://github.com/thorade/HelmholtzMedia/>.
- [10] M. Thorade and A. Saadat. “Partial derivatives of thermodynamic state properties for dynamic simulation”. In: *will be submitted to: Environmental Earth Sciences* (2012).
- [11] H. Tummescheit. *Ticket 85: Re-design and simplification of Modelica.Media*. 2008. URL: <https://trac.modelica.org/Modelica/ticket/85>.
- [12] W. Wagner. *Eine mathematisch statistische Methode zum Aufstellen thermodynamischer Gleichungen — gezeigt am Beispiel der Dampfdruckkurve reiner fluider Stoffe*. Vol. 3. Fortschrittberichte der VDI Zeitschriften 39. VDI Verlag, 1974.
- [13] W. Wagner and A. Pruß. “The IAPWS Formulation 1995 for the Thermodynamic Properties of Ordinary Water Substance for General and Scientific Use”. In: *Journal of Physical and Chemical Reference Data* 31.2 (2002), pp. 387–535. DOI: 10.1063/1.1461829.

Object-Oriented Library of Switching Moving Boundary Models for Two-phase Flow Evaporators and Condensers

Javier Bonilla ^a Luis J. Yebra ^a Sebastián Dormido ^b François E. Cellier ^c
^a Centro de Investigaciones Energéticas MedioAmbientales y Tecnológicas (CIEMAT)
Plataforma Solar de Almería (PSA), Almería, Spain
^b National Distance Education University (UNED),
Department of Computer Science and Automatic Control, Madrid, Spain
^c Swiss Federal Institute of Technology (ETH Zurich),
Department of Computer Science, Zurich, Switzerland

Abstract

This paper discusses a Modelica library of switching moving boundary models for two-phase flow heat exchangers: evaporators and condensers. The equation-based object-oriented modeling paradigm has been considered by means of designing basic models applying the conservation laws for each flow state: subcooled liquid, two-phase flow and superheated vapor. Evaporator and condenser models have been developed by interconnecting the basic models and including mechanisms to switch between different configurations: general, flooded and dry evaporators and condensers. Finally, simulation results are presented by an integrity and stability test case.

Keywords: Moving boundary model; switching; two-phase flow; evaporator; condenser

1 Introduction

Heat exchangers play a very important role in industry; the modeling and control of these elements is a key part in the process plant control. Two of the most common discretization approaches used in fluid dynamic modeling are the finite-volume distributed-parameter method [21] and the moving-boundary lumped-parameter method [8]. Dynamic modeling is always a challenging task in which the trade-off between accuracy and speed must be evaluated depending on the purpose of the model. Moving boundary models are low-order and much faster models than finite volume models; additionally they can describe the dynamic behavior of evaporators and condensers with high accuracy [1]. In the context of real-time simulation, dynamic system optimization and model-based

control, where fast computation is required, the moving boundary method seems to be appropriate.

The moving boundary method divides the evaporator/condenser in different regions, also called Control Volumes (CVs), depending on the fluid phase. In each CV, the lumped thermodynamic properties are averaged; the barrier is not fixed and it may move between adjacent CVs. The main idea is to dynamically track the lengths of the different regions [16].

The three basic flow states are: subcooled liquid (SC), two-phase flow (TP) consisting of vapor and liquid present simultaneously in the same volume, and superheated vapor (SH) as represented in Fig. 1. Considering these three basic flow states, compound configurations can be created. Fig. 2 shows these configurations: general, flooded and dry evaporators/condensers.

A state-of-the-art study in moving boundary models for two-phase flow heat exchangers was previously presented in [2] together with a new switching flooded evaporator model. This paper extends previous work by new switching moving boundary models for general/dry evaporators and general/flooded/dry condensers. To the knowledge of the authors, there are three papers related to moving boundary models developed using Modelica [17, 27, 13]. The novelty of this paper is that a strictly object-oriented design is followed.

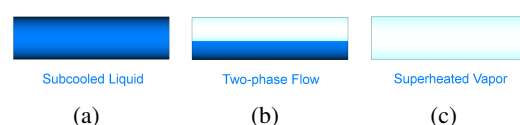


Figure 1: Basic flow states

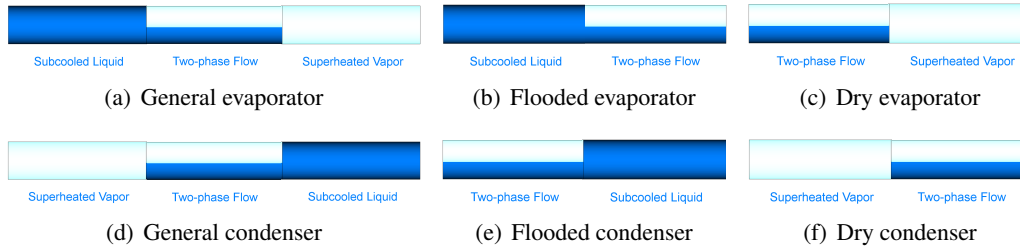


Figure 2: Evaporator and condenser configurations

2 Mathematical modeling

This section first describes the assumptions made in the development of the mathematical models, after that the governing equations in their general form are listed, the one-dimensional CV concept is then introduced, and finally the basic and compound models are explained together with some additional equations required to complete the models.

2.1 Assumptions

With the aim of developing a low-order model that reflects the principal dynamics, a number of assumptions have been made: horizontal orientation; one-dimensional case; constant pipe cross-sectional area; time-dependent uniform pressure along the evaporator; homogeneous two-phase flow; average properties and time-dependent uniform heat flux per unit length in each CV; negligible gravitational forces; negligible changes in the kinetic energy; negligible viscous stress; heat conduction and radiation in the fluid and heat conduction in the pipe wall are also neglected.

2.2 Governing equations

The straightforward way to derive the model equations is from the time-dependent equations for conservation laws. Considering the assumptions presented in the previous section, the differential formulation for the conservation of mass and energy in the fluid are represented by Eqs. 1 and 2, respectively [18]. Eq. 3 [18] defines the conservation of energy in the pipe wall and Table 1 summarizes the nomenclature.

$$\frac{\partial A\rho}{\partial t} + \frac{\partial \dot{m}}{\partial z} = 0, \quad (1)$$

$$\frac{\partial A\rho u}{\partial t} + \frac{\partial \dot{m}h}{\partial z} = \dot{q}_i, \quad (2)$$

$$A_w \rho_w c_{p,w} \frac{\partial T_w}{\partial t} = \dot{q}_o - \dot{q}_i. \quad (3)$$

Var.	Description	Units	
t	Time	[s]	
z	Spatial coordinate	[m]	
A	Cross-sectional area	[m ²]	
c_p	Isobaric specific heat capacity	[J/(K·kg)]	
\dot{m}	Mass flow rate	[kg/s]	
x	Vapor quality	[-]	
p	Pressure	[Pa]	
\dot{Q}	Heat flow rate	[W]	
\dot{q}	Heat flux	[W/m ²]	
γ	Void fraction	[-]	
$\bar{\gamma}$	Mean void fraction	[-]	
h	Specific enthalpy	[J/kg]	
\bar{h}	Mean specific enthalpy	[J/kg]	
h'	h of saturated liquid	[J/kg]	
h''	h of saturated vapor	[J/kg]	
ρ	Density	[kg/m ³]	
$\bar{\rho}$	Mean density	[kg/m ³]	
ρ'	Density of saturated liquid	[kg/m ³]	
ρ''	Density of saturated vapor	[kg/m ³]	
T	Temperature	[K]	
\bar{T}	Mean temperature	[K]	
ε	Pipe roughness	[m]	
Subs.	Description	Subs.	Description
a	Inlet to CV	b	Outlet to CV
sc	Subcooled	tp	Two-phase
sh	Superheated	w	Pipe wall
i	Inner to CV	o	Outer to CV

Table 1: Nomenclature

2.3 One-dimensional Control Volume

The moving boundary method is based on the division of the heat exchanger in different CVs. Fig. 3 represents a CV; the lumped thermodynamic properties in the CV are averaged and they are uniform but time-dependent ($\bar{h}, \bar{T}, \bar{\rho}$); the pressure (p) is not denoted by a mean value, because there is only one time-dependent pressure value for the entire evaporator. The cross-sectional areas (A, A_w) are constant. Each CV has three interfaces or boundaries. One is adjacent to the pipe wall where the thermodynamic properties are also considered in its mean values ($\bar{T}_w, \bar{\rho}_w$).

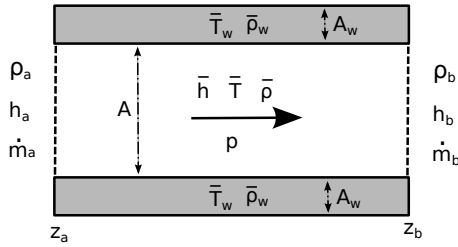


Figure 3: Control Volume (CV)

The other two interfaces connect to adjacent CVs or the inlet or outlet boundaries of the heat exchanger. In Fig. 3 the flow direction is defined by the arrow, so the inlet flow thermodynamic properties correspond to the a subscript variables (ρ_a, h_a, \dot{m}_a), whereas the outlet flow thermodynamic properties are defined by the b subscript variables (ρ_b, h_b, \dot{m}_b).

2.4 Basic Volume Models

The derivation of the mass and energy balance equations for the CV models is not presented due to space limitation. From the state-of-the-art study in two-phase flow moving boundary models useful information was obtained [2]. The derivation of the model is analogous to the developed in [16, 17] but not neglecting the mean void fraction time derivative ($d\bar{\gamma}/dt$), where a new calculation method has been introduced. Additionally, the thermodynamic properties at the boundaries are not fixed to any particular value, by means of considering the density or specific enthalpy of saturated liquid/vapor, so the basic volume models can be used in any evaporator/condenser.

2.4.1 One-phase Flow Volume Model

The mass and energy balance equations for the subcooled liquid and superheated vapor CV models are described by Eqs. 4 and 5 where the subscript cv can be substituted by sc or sh to consider the particular CV.

$$A \left(z_{cv} \frac{d\bar{\rho}_{cv}}{dt} + \bar{\rho}_{cv} \frac{dz_{cv}}{dt} \right) + \rho_a A \frac{dz_a}{dt} - \rho_b A \frac{dz_b}{dt} = \dot{m}_a - \dot{m}_b. \quad (4)$$

$$A \left(\bar{\rho}_{cv} \bar{h}_{cv} \frac{dz_{cv}}{dt} + \bar{\rho}_{cv} \frac{d\bar{h}_{cv}}{dt} z_{cv} + \frac{d\bar{\rho}_{cv}}{dt} \bar{h}_{cv} z_{cv} \right) - A z_{cv} \frac{dp}{dt} + A \rho_a h_a \frac{dz_a}{dt} - A \rho_b h_b \frac{dz_b}{dt} = \dot{m}_a h_a - \dot{m}_b h_b + \dot{q}_{i,cv} z_{cv}. \quad (5)$$

2.4.2 Two-phase Flow Volume Model

The mass and energy balance equations for the two-phase flow CV model are described by Eqs. 6 and 7. The way the mean void fraction and its time derivative are calculated is described in [2].

$$A \left(\frac{dz_{tp}}{dt} (\bar{\gamma} \rho'' + (1 - \bar{\gamma}) \rho') + z_{tp} \left(\frac{d\bar{\gamma}}{dt} (\rho'' - \rho') + \bar{\gamma} \frac{d\rho''}{dp} \frac{dp}{dt} + (1 - \bar{\gamma}) \frac{d\rho'}{dp} \frac{dp}{dt} \right) \right) + \rho_a A \frac{dz_a}{dt} - \rho_b A \frac{dz_b}{dt} = \dot{m}_a - \dot{m}_b. \quad (6)$$

$$A \left(\frac{dz_{tp}}{dt} (\bar{\gamma} \rho'' h'' + (1 - \bar{\gamma}) \rho' h') + z_{tp} \left(\frac{d\bar{\gamma}}{dt} (\rho'' h'' - \rho' h') + \bar{\gamma} \frac{d\rho''}{dp} \frac{dp}{dt} h'' + \bar{\gamma} \rho'' \frac{dh''}{dp} \frac{dp}{dt} + (1 - \bar{\gamma}) \frac{d\rho'}{dp} \frac{dp}{dt} h' + (1 - \bar{\gamma}) \rho' \frac{dh'}{dp} \frac{dp}{dt} \right) \right) - A z_{tp} \frac{dp}{dt} + A \rho_a h_a \frac{dz_a}{dt} - A \rho_b h_b \frac{dz_b}{dt} = \dot{m}_a h_a - \dot{m}_b h_b + \dot{q}_{i,tp} z_{tp}. \quad (7)$$

2.5 Heat Exchanger Models

When modeling the compound models (not only one CV model), additional equations are required besides the CV governing equations; these equations depend on the kind of heat exchanger and relate the outlet CV specific enthalpies with the values at saturation conditions.

2.5.1 Evaporator

If a general or flooded evaporator is considered (cf. Figs. 2(a) and 2(b)), Eq. 9(a) is required for the subcooled liquid CV, and also the initial value for h_b must be set to h' .

An easy way to accomplish this is to only introduce Eq. 8. However, there is a problem with that approach if switching moving boundaries models are considered.

$$h_b = h'. \quad (8)$$

Suppose that a flooded evaporator is being modeled, where the outlet fluid is two-phase flow; h_b for the subcooled liquid CV is not a state variable because it depends on pressure, and therefore Eq. 8 is valid

and h_b is an algebraic variable. However, if the outlet fluid turns into subcooled liquid due to a change in the model inputs, Eq. 8 is no longer valid and h_b is a state variable. Such a model is called a *variable-structure model*. In a variable-structure model the number/type of equations or variables can change, on the other hand a static-structure model implies that the number of equations as well as the number of algebraic and state variables remains the same. Variable-structure models are not currently supported by most modeling and simulations tools (including Modelica tools). Whereas there exist some modeling languages and tools that support variable-structure models, none of the existing variable-structure modeling tools supports the handling of higher-index systems [28]. For that reason, Modelica is still our preferred modeling language, but it must be taken into account that only static-structure models can be simulated.

For this reason, the number of equations must remain the same in all different configurations of our model, and h_b for the different CVs must always be a state variable so its value cannot be fixed to any algebraic variable and neither can h_a , because it is connected to h_b from the CV to the left, except for the case of the first CV where the h_a value can be freely establish.

If a general or dry-expansion evaporator is considered (cf. Figs. 2(a) and 2(c)), Eq. 9(b) is required for the two-phase flow CV, and also the initial value for h_b must be set to h'' .

2.5.2 Condenser

If a general or flooded condenser is considered (cf. Figs. 2(d) and 2(e)), Eq. 9(a) is required for the two-phase flow CV, and also the initial value for h_b must be set to h' . If a general or dry condenser is considered (cf. Figs. 2(d) and 2(f)), Eq. 9(b) is required for the superheated vapor CV, and also the initial value for h_b must be set to h'' .

$$\frac{dh_b}{dt} = \frac{dh'}{dt} \quad (a), \quad \frac{dh_b}{dt} = \frac{dh''}{dt} \quad (b). \quad (9)$$

2.6 Pipe Wall Model

The energy balance equation for each pipe wall CV is described by Eq. 10. This equation is derived in [27], where $T_{w,a}$ and $T_{w,b}$ are the wall temperature values at the interfaces. This approach is closer to the practical situation as it remains continuous and smooth during

the switching between different configurations.

$$A_w \rho_w c_{p,w} \left(\frac{d\bar{T}_w}{dt} + \frac{\bar{T}_w - T_{w,b}}{z_{ab}} \frac{dz_b}{dt} + \frac{T_{w,a} - \bar{T}_w}{z_{ab}} \frac{dz_a}{dt} \right) = \dot{q}_o - \dot{q}_i. \quad (10)$$

2.7 Additional Equations

Some additional equations are required in order to complete the heat exchanger model. These equations are not detailed here due to space limitations but they can be easily found in the literature [16]. The remaining equations are: the heat flow rates between the pipe wall and the ambient and between the pipe wall and the fluid and the geometric constraints, i.e., the total heat exchanger length and the pipe geometry. The pipe geometry considered in this manuscript has been the cylindrical geometry.

2.8 Switching

Switching from one configuration to another implies the disappearance of an existing CV or the appearance of a new one, e.g. when switching from a general evaporator to a flooded evaporator or vice versa. This section elaborates how such transitions are captured by the model. Additional equations for the new CV may be required. When the CV is active, its governing equations correspond to the equations described in Sections 2.4.1 or 2.4.2 depending on the fluid phase; however a different set of equations is required to describe the CV in its inactive state. This is also explained in this section. It is assumed that the appearance or disappearance of a CV can only occur at the end of the heat exchanger.

2.8.1 Disappearance of a Control Volume

A CV disappears (becomes inactive) when Eq. 11(a) becomes true, where z_{min} denotes a threshold that specifies the minimum length of an active CV. This value cannot be zero in order to avoid structural singularities, therefore the CV length must be greater than zero. The default value for this parameter has been set to 10^{-6} m.

2.8.2 Control Volume in an Inactive State

When any of the CVs is inactive, the mass and energy balance equations (Eqs. 4 and 5 or Eqs. 6 and 7 depending on the CV fluid phase) are substituted by

Eqs. 11(b) and 11(c), respectively. These equations guarantee that the CV is inactive and does not act on the fluid.

$$z_{cv} < z_{min} \quad (a), \quad \dot{m}_a = \dot{m}_b \quad (b),$$

$$\frac{dh_a}{dt} = \frac{dh_b}{dt} \quad (c), \quad \frac{dz_{cv}}{dt} = 0 \quad (d). \quad (11)$$

Moreover, Eq. 9(a) or 9(b) must be substituted by Eq. 11(d) depending on the inactive CV and on the kind of heat exchanger considered.

2.8.3 Appearance of a Control Volume

The event triggering the appearance of a CV depends on the particular CV and also on the kind of heat exchanger.

Evaporator. The superheated vapor CV appears (cf. Figs. 2(a) and 2(c)) when the vapor quality in the two-phase flow CV becomes greater than 1.0, Eq. 12(a). The two-phase flow CV appears (cf. Figs. 2(a) and 2(b)) when the outlet specific enthalpy in the subcooled liquid CV becomes greater than the specific enthalpy of saturated liquid, Eq. 12(b).

Condenser. The subcooled liquid CV appears (cf. Figs. 2(d) and 2(e)) when the outlet specific enthalpy in the two-phase flow CV is lower than the specific enthalpy of saturated liquid, Eq. 12(c). The two-phase flow CV appears (cf. Figs. 2(d) and 2(f)) when the outlet specific enthalpy in the superheated vapor CV becomes lower than the specific enthalpy of saturated vapor, Eq. 12(d).

$$x > 1 \quad (a), \quad h_b > h' \quad (b),$$

$$h_b < h' \quad (c), \quad h_b < h'' \quad (d). \quad (12)$$

3 Description of the Library

This section describes the Modelica library that implements the mathematical models previously described, the *MBMs (Moving Boundary Models) library*.

3.1 Library Structure and Interfaces

Fig. 4(a) shows the main packages that make up the MBMs library, and Fig. 4(b) shows the *Components.Water.MBM* package in expanded view, where the basic and compound models can be seen. The

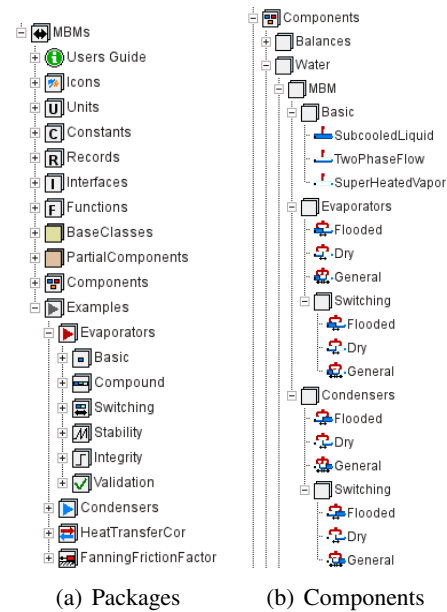


Figure 4: The MBMs library

former do not support switching, whereas the latter do. *Modelica Fluid* and *Modelica Thermal* ports have been used throughout in order to define the interfaces in the MBMs library. This guarantees that the MBMs library is compatible with any component from the *Modelica standard library 3.2* [20] or from third-party components that also make use of these interfaces.

3.2 Partial Base Classes

The most remarkable partial base classes in the MBMs library are: the Volume class, the MultipleVolume class, the HeatTransferCorrelation class and the FrictionFactor class.

3.2.1 Volume Class

The Volume class defines the fluid and heat ports, the medium, some additional thermodynamic properties, as well as the state and geometry of the CV. This class is the base class for the basic volume models. The volume class also includes a heat transfer correlation (HTC) and a friction factor model (FFM).

3.2.2 MultipleVolume Class

The MultipleVolume class defines two or three CVs that can be redeclared in classes that inherit from it. The CVs are connected through the fluid connectors, and this is the base class for all heat exchangers. We

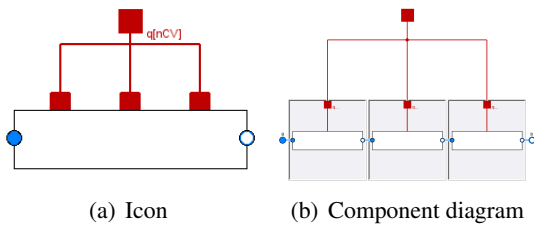


Figure 5: MultipleVolume base class (3 CVs)

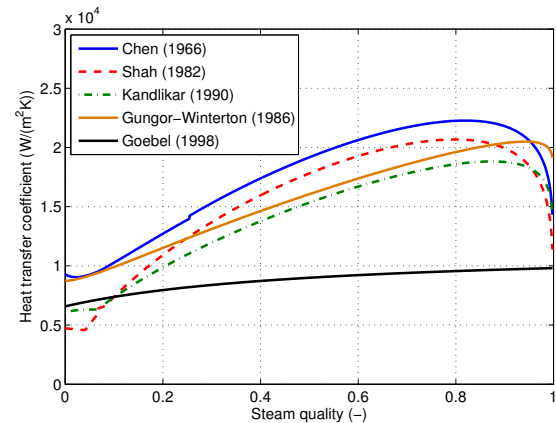
followed strictly an object-oriented design for heat exchangers. Figs. 5(a) and 5(b) show the icon and component diagram of the 3 CVs MultipleVolume class.

3.2.3 Heat Transfer Correlations and Friction Factor Models

There are two base classes for heat transfer correlations (HTCs) and friction factor models (FFMs). The user can inherit from them to define new HTCs or FFMs. FFMs have been implemented because the Petukhov and Gnielinski HTCs require a friction factor which can be calculated from those FFMs, furthermore there are plans for extending the library with pressure loss. A HTC can be restricted to only one particular fluid phase (one-phase or two-phase) or to only one particular process (evaporation or condensation), if required. Moreover, there are some HTCs for evaporation and FFMs for smooth and rough pipes, always considering turbulent flow, already implemented in the library. They are summarized in Table 2. The implemented HTCs and FFMs have been also adapted to switching in order to avoid discontinuities and numerical problems during the simulation. The HTCs and FFMs have been validated against an independent implementation [26]. The HTC and FFM can be selected in each CV through the GUI. A test case for the implemented two-phase flow HTCs is shown in Fig. 6.

3.3 Volume Components

Fig. 7 shows the icons of the subcooled liquid, two-phase flow and superheated vapor models. These models inherit from the Volume class and add their particular equations, although the subcooled liquid and the superheated vapor models inherit from an intermediate class in the hierarchy, the OnePhaseVolume class, because both models share the same equations.


 Figure 6: Comparison of two-phase flow HTCs ($p = 3$ MPa, $\dot{Q} = 5,827$ Kw, $\dot{m} = 0.6$ kg/s, $\varepsilon = 3 \cdot 10^{-5}$ m)

Heat Transfer Correlations	Fluid phase
Ideal	any
Constant	any
[9] Dittus-Boelter (1930)	One-phase
[3] Chen (1966)	Two-phase
[22] Petukhov (1970)	One-phase
[11] Gnielinski (1976)	One-phase
[25] Shah (1982)	Two-phase
[14] Gungor-Winterton (1986)	Two-phase
[19] Kandlikar (1990)	Two-phase
[12] Goebel (1998)	Two-phase
Fanning Friction Factor Model	Kind of pipe
None	-
Constant	any
[5] Colebrook (1939)	any
[4] Chen (1979)	any
Explicit simplified Chen (1979)	any
[24] Karman-Prandtl (1930)	Rough
[7] Denn (1980)	Smooth
[15] Haaland (1983)	any

Table 2: HTCs for evaporation and fanning FFMs implemented in the MBMs library

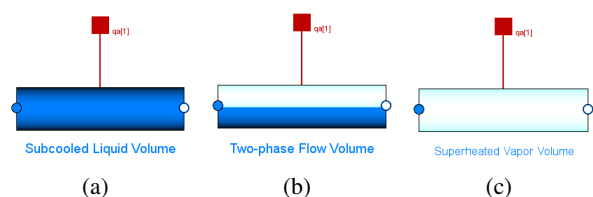


Figure 7: Volume components

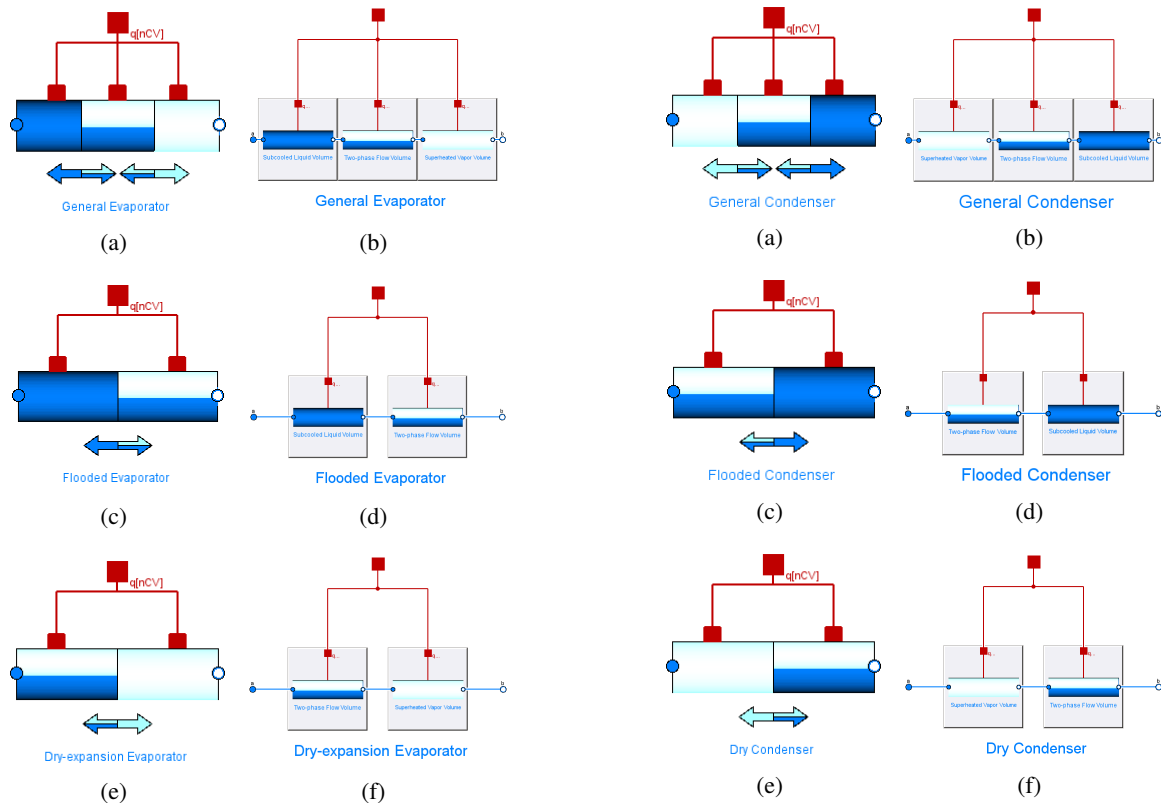


Figure 8: Evaporator components

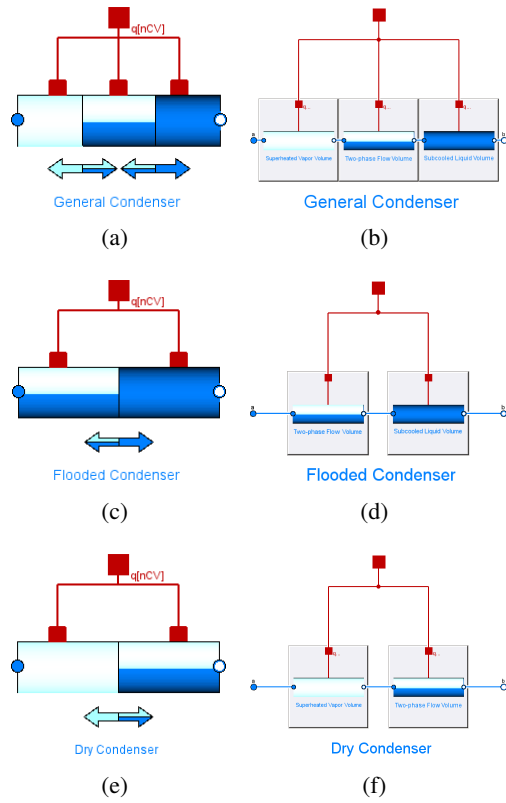


Figure 9: Condenser components

3.4 Heat Exchanger Components

Redeclaring the partial Volume classes in the MultipleVolume (2 or 3 CVs) class with the volume components: subcooled liquid, two-phase flow and superheated vapor models, evaporators and condenser can be defined.

3.4.1 Evaporator Components

Fig. 8 shows the general, flooded and dry-expansion evaporator models. The figures on the left represent the icons whereas the figures on the right are the component diagrams, where the partial Volume classes have been redeclared appropriately.

3.4.2 Condenser Components

For condensers, the situation is the same, but changing the order of the interconnected basic volumes models. Fig. 9 shows the general, flooded and dry condenser icons and component diagrams.

3.5 Pipe Wall Component

The pipe wall component includes the pipe wall model previously introduced in Section 2.6 adapted to support switching. The pipe wall component together with the Volume class depend on the geometry. The geometry is a partial class. Different geometries can be implemented by inheriting from the geometry class; the library already includes a cylindrical geometry model. Moreover, the pipe wall component inherits from a partial wall class, so different wall models can be implemented and used.

3.6 The initialization problem

The initialization problem is always a cumbersome task and it is especially difficult when considering inactive CVs in the initialization. For that reason, the initialization has been taken into account in the design of the MBMs library; the initialization options can be established through the GUI in the initialization tab of evaporators and condensers. Fig. 10 shows the initialization options for a switching general evaporator. Here, it can be specified, which CVs are inactive in the initialization, the initial inlet pressure can be set as well as the initial outlet temperature (this value is

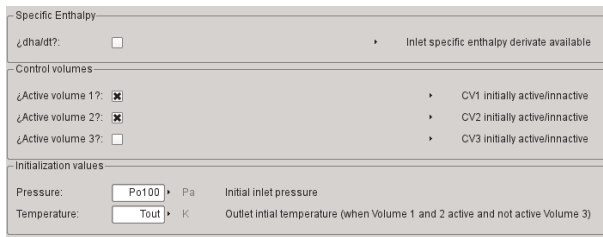


Figure 10: Initialization options for general evaporators

only required when the outlet fluid is two-phase flow), and it can be specified whether the inlet specific enthalpy time derivative is available. Sometimes when considering experimental data as input, this thermodynamic property may not be available and cannot be estimated, in which case the inlet specific enthalpy time derivative is set to zero.

4 Simulation

This section shows the simulation of the mathematical models previously introduced and implemented in the MBMs library. The medium in these simulations is the two-phase flow water-steam mixture from the *Modelica Media library* [20]. Dymola 2013 [6] has been the Modelica tool used for these simulations. The numerical solver used has been DASSL [23] and the relative tolerance has been set to 10^{-6} . All of the developed models have been thoroughly tested in integrity and stability tests, however due to space limitation only a few can be presented in this article. A simulation test for a switching flooded evaporator was presented in [2].

4.1 Model Integrity

The simulation results must be verified and the governing equations of the model must be validated both in steady-state and in transient predictions. To this end, the mathematical model and library implementation results were compared to those of an independently developed finite volume model and code from the *Modelica Fluid library* [10] that belongs to the *Modelica Standard Library 3.2*. The *Modelica Fluid library* has been meticulously designed and tested and is widely used in the Modelica community.

Fig. 11 shows the outlet temperature for a test case considering a switching moving boundary general evaporator model from the MBMs library (dashed

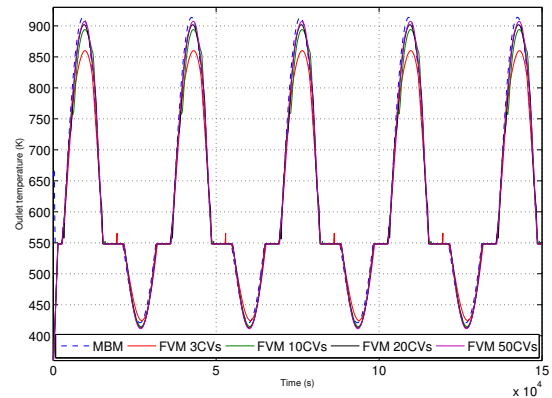


Figure 11: Integrity and stability test

Model	CPU-Time (s)	State events
MBM	0.87	104
FVM 3 CVs	2.34	45
FVM 10 CVs	6.93	124
FVM 20 CVs	21.4	228
FVM 50 CVs	103	551

Table 3: Simulation statistics

blue line) and finite volume models from the *Modelica Fluid library* considering different numbers of CVs (3,10,20,50). It can be seen that the simulation results obtained with the MBMs library are in good agreement with those from the *Modelica Fluid library* and that the MBMs library model runs considerably faster (cf. Table 3), because the finite volume model requires at least 20 CVs to yield acceptably accurate results.

4.2 Model Stability

Model stability, especially the switching stability, was checked by holding certain inputs constant during the simulation while varying sinusoidally others to force repeated switching. Variations in the heat flow rate, mass flow rate, inlet specific enthalpy and outlet pressure have been tested. Fig. 11 shows the outlet temperature in a switching general evaporator when varying sinusoidally the inlet heat flow rate over the pipe (cf. Fig. 12). The outlet fluid phase changes from subcooled liquid to two-phase flow (constant temperature) to superheated vapor. Fig. 13 shows the CV lengths in the moving boundary model where it can be observed, which CVs are inactive during the simulation (CVs with zero length), the length of the evaporator is 500 m.

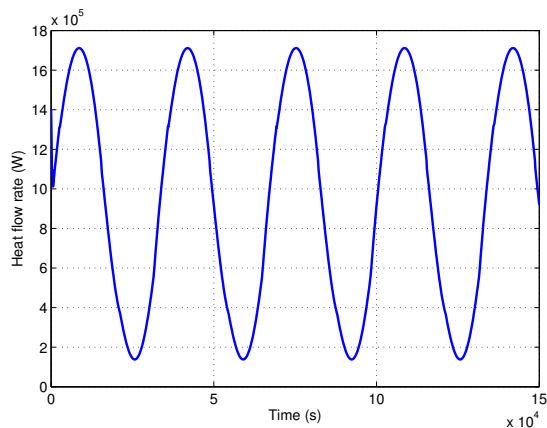


Figure 12: Heat flow rate over the pipe

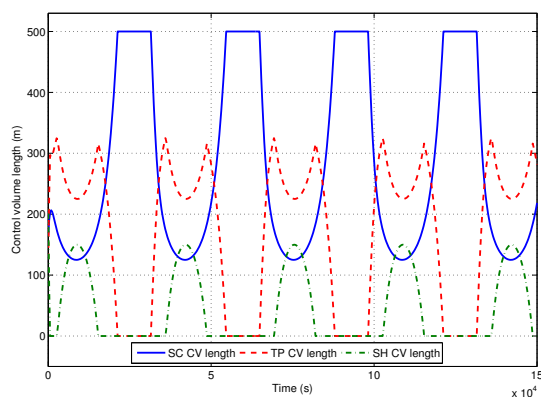


Figure 13: Control volume lengths

5 Conclusions

This paper details mathematical moving boundary models for heat exchangers, considering basic CVs and compound models: general, flooded and dry evaporators and condensers, independent of the two-phase flow medium. The pipe wall model is also shown. It is independent of the geometry, particularized for a cylindrical geometry in this paper. The switching criteria was also introduced allowing the disappearance of the CVs at the end of the heat exchanger. A new equation-based object-oriented Modelica library, the MBMs library, implementing all of the detailed models has been presented. This library provides models of different HTC and FFM. It also tackles the initialization problem, which is specially tough in the case of moving boundary models. The mathematical models and the MBMs library have been tested thoroughly using integrity and stability tests.

6 Future work

The MBMs library is currently still in its beta version, and some of the following open tasks will be considered for future library extensions: pressure drop in each CV and disappearance of CVs at the beginning of the heat exchanger. It is planned to use and validate the switching condenser models in the modeling of a double effect absorption heat pump in the ambit of the *POWER* project. The switching evaporator models are intended to be also validated in the *HIBIOSOLEO* project for the development of a direct steam generation linear Fresnel solar thermal power plant.

References

- [1] S. Bendapudi, J. Braun, and E. Groll. A comparison of moving-boundary and finite-volume formulations for transients in centrifugal chillers. *International Journal of Refrigeration*, 31(8):1437–1452, December 2008.
- [2] J. Bonilla, L.J. Yebra, S. Dormido, and F.E. Cellier. Object-Oriented Modeling of Switching Moving Boundary Models for Two-phase Flow Evaporators. In *Proceedings MATHMOD 2012 - 7th Vienna International Conference on Mathematical Modelling*, 2012.
- [3] J.C. Chen. Correlation for Boiling Heat Transfer to Saturated Fluids in Convective Flow. *Industrial Engineering Chemistry Process Design and Development*, 5(3):322–329, 1966.
- [4] N.H. Chen. An Explicit Equation for Friction Factor in Pipe. *Industrial & Engineering Chemistry Fundamentals*, 18(3):296–297, August 1979.
- [5] C.F. Colebrook. Turbulent Flow in Pipes, with particular reference to the Transition Region between the Smooth and Rough Pipe Laws. *Journal of the Institution of Civil engineers*, 11(4):133–156, 1939.
- [6] Dassault Systèmes. Dymola 2013 - Dynamic Modeling Laboratory. <http://www.3ds.com/products/catia/portfolio/dymola>, 2012.
- [7] M.M. Denn. *Process Fluid Mechanics*. Number 6. Prentice-Hall, Englewood Cliffs, 1980.
- [8] M. Dhar and W. Soedel. Transient Analysis of a Vapor Compression Refrigeration System. In

- Proceedings of the 15th International Congress of Refrigeration*, pages 1031 – 1067, Venice, Italy, 1979.
- [9] 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.
- [10] R. Franke, F. Casella, M. Sielemann, K. Proelss, M. Otter, and M. Wetter. Standardization of Thermo-Fluid Modeling in Modelica.Fluid. In *Proc. of the 7th Int. Modelica Conference*, pages 122–131, Italy, September 2009.
- [11] V. Gnielinski. New equations for heat and mass transfer in turbulent pipe flow and channel flow. *International Chemical Engineering*, 2(16):359–368, 1976.
- [12] O. Goebel. Thermohydraulics of Direct Steam Generation. In *Proceedings of the 9th International Symposium on Solar Thermal Concentrating Technologies*, Odeillo-Font-Romeu, 1998.
- [13] M. Gräber, N.C. Strupp, and W. Tegethoff. Moving Boundary Heat Exchanger Model and Validation Procedure. In *Proceeding of EUROSIM*, 2010.
- [14] E. Gungor and R.H.S. Winterton. A general correlation for flow boiling in tubes and annuli. *International Journal of Heat and Mass Transfer*, 29(3):351–358, 1986.
- [15] S.E. Haaland. Simple and Explicit Formulas for the Friction Factor in Turbulent Pipe Flow. *Journal of Fluids Engineering*, 105(1):89–90, 1983.
- [16] J.M. Jensen. *Dynamic modeling of Thermo-Fluid Systems-With focus on evaporators for refrigeration*. PhD thesis, Technical University of Denmark, 2003.
- [17] J.M. Jensen and H. Tummescheit. Moving boundary models for dynamic simulations of two-phase flows. In *Proc. of the 2nd Int. Modelica Conference*, 2002.
- [18] R.W. Johnson. *The Handbook of Fluid Dynamics*. CRC Press, 1998.
- [19] S.G. Kandlikar. A general correlation for saturated two-phase flow boiling heat transfer inside horizontal and vertical tubes. *Journal of heat transfer*, 112:219 – 228, 1990.
- [20] Modelica Association. Modelica Standard Library 3.2, 2010.
- [21] S.V. Patankar. *Numerical Heat Transfer and Fluid Flow*. Hemisphere, Washington,D.C, 1980.
- [22] B.S. Petukhov. Heat Transfer and Friction in Turbulent Pipe Flow with Variable Physical Properties. *Advances in Heat Transfer*, 6(C):504–564, 1970.
- [23] L.R. Petzold. A description of DASSL: a Differential/Algebraic System Solver. *Scientific Computing*, pages 65–68, 1983.
- [24] H. Schlichting and K. Gersten. *Boundary-layer theory*. Springer, 2000.
- [25] M.M. Shah. Chart correlation for saturated boiling heat transfer: equations and further study. *ASHRAE TransUnited States*, 88(CONF-820112-):185–196, 1982.
- [26] E. Zarza. *The Direct Steam Generation with Parabolic Collectors. The DISS project (in Spanish)*. PhD thesis, Escuela Superior de Ingenieros Industriales de Sevilla, Seville, Spain, November 2000.
- [27] W. Zhang and C. Zhang. A generalized moving-boundary model for transient simulation of dry-expansion evaporators under larger disturbances. *International Journal of Refrigeration*, 29(7):1119–1127, November 2006.
- [28] D. Zimmer. *Equation-Based Modeling of Variable-Structure Systems*. PhD thesis, Swiss Federal Institute of Technology (ETH), 2010.

Acknowledgments

This work has been financed by CIEMAT research centre, by the INNFACTO project, *Hibridación de tecnologías renovables en una planta de generación de energía. (HIBIOSOLEO)*, IPT-440000-2010-004 and the National Plan Project, *Predictive COntrol techniques for efficient management of reneWable Energy micro-gRids. (POWER)*, DPI2010-21589-C05-02 of the Spanish Ministry of Economy and Competitiveness and FEDER funds.

High-Speed Compressible Flow and Gas Dynamics

Michael Sielemann

Deutsches Zentrum für Luft- und Raumfahrt, Robotics and Mechatronics Center,
System Dynamics and Control, Münchner Strasse 20, 82234 Wessling, Germany

Abstract

Discretization schemes suitable for gas dynamics are reviewed and applied to the declarative concepts of Modelica. Here, a suitable connector definition is introduced to enable both robust simulation and higher-order schemes, which require larger stencils than typically available on established thermo-fluid dynamics connectors.

Keywords: Finite volume method, shock waves, monotone flux, total variation diminishing, essentially non-oscillatory

1 Introduction

System-level simulation of thermo-fluid dynamics using Modelica is a wide topic yet relatively mature. Several authors present applications using the language in various technical domains. For instance, Casella [3, 4] considers power plant simulation, Pfafferott [20], Tummescheit et al. [36], Richter [24], and Pröhl [21] study applications in sub-critical vapor compression cycles, Casas [2, 1] addresses air conditioning using desiccant assisted cycles, and Vasel and Schmitz [40] consider air conditioning using trans-critical cycles.

In all of the given applications, the governing equations are adapted to the specifics of the underlying flow phenomena. With the exception of López [5], the assumptions are identical for all applications reported in literature. The corresponding flow, which allows to make these assumptions, is called a *low-speed compressible flow* herein. All authors referenced in the first paragraph assume that the flow is incompressible with respect to the flow phenomena, as it is low-speed. Density variation is only encountered due to heat transfer and in lumped parameter components such as compressors. Density variation due to flow phenomena is neglected, i.e., the Mach number is typically below 0.3.

In particular, an analysis of model code revealed that the difference between static and total pressure is neglected as the dynamic pressure is considered small and not of interest. For the given applications in power plants or vapor compression cycle refrigeration systems this is reasonable. Only in special devices, which involve large variations in flow cross-section such as adapters between different pipe diameters or nozzles, total pressure is of interest. Total or stagnation enthalpy is often treated similarly; the kinetic term $v^2/2$ is neglected. A typical argument is that the order of magnitude of change in specific enthalpy due to heat transfer is larger than that of such kinetic terms.

If kinetic terms in pressure and specific enthalpy are not neglected for such applications and the common assumption of a steady-state momentum balance is made then coupled nonlinear algebraic equation systems arise for density, which is required to establish flow velocity. These coupled equation systems deteriorate simulation performance.

Certain applications involve a different type of flow, which is called *high-speed compressible flow* herein. Kinetic terms and dynamic pressure may not be neglected and have to be included in compressible formulations. Density variation is also encountered with respect to flow phenomena, in particular dynamic conservation of momentum is relevant and also shock waves may be part of the solution. The Mach number may be > 0.3 (including the supersonic regime). The term “gas dynamics” refers to the same type of flow.

The key theoretical area to enable applications involving high-speed compressible flow is the discretization method for the governing equations. The foundations of numerical solution methods in thermo-fluid dynamics are well understood. However, in the framework of equation-based, object-oriented modeling languages, only methods suitable for low-speed compressible flow have been applied. The classic finite volume method has been studied in par-

ticular by Tummescheit [35]. Moving boundary methods have been applied by Jensen [14, 15] and Tummescheit [35]. Casella [4] proposed a mean density discretization, which is non-conservative but results in continuous and continuously differentiable thermodynamic properties at phase boundaries of two-phase flow. Pröhl and Schmitz [22] discretized the governing equations for frost formation on heat exchanger surfaces.

López [5] proposed an approach to model and simulate gas dynamics. Due to robustness issues, which are certainly linked to deficiencies in the connector definition used in [5] (c.f. reference [7]), the approach did not become widely supported. In an attempt to finally extend the applicability of Modelica also to high-speed compressible flow and gas dynamics, this paper and reference [29] contribute to the state of the art in the following areas.

- Relevant concepts of the theory in numerical solution methods for high-speed compressible flow are reviewed and translated from the algorithmic perspective taken in literature to the acausal concepts of equation-based, object-oriented modeling languages.
- The elements of discretization schemes are decomposed in an object-oriented fashion and implemented in a generic library. Object-oriented concepts are exploited for increased flexibility such as parametric polymorphism for exchangeable thermodynamic property models.

2 The governing equations in compact flux form

To address high-speed compressible flow, a compact flux formulation as described by Toro [34] is considered. It is posed using conserved variables u and flux f .

$$u_t(x,t) + f(u(x,t))_x = s(u(x,t)) \quad (1)$$

$$u(x,t) = \begin{pmatrix} \rho \\ \rho v \\ \rho u_0 \end{pmatrix} \quad (2)$$

$$f(u(x,t)) = \begin{pmatrix} \rho v \\ \rho v^2 + p \\ v(\rho u_0 + p) \end{pmatrix} \quad (3)$$

If the cross-sectional area A is supposed to vary smoothly with time and position, then the following

source term including heat transfer and viscous wall friction can be used [34].

$$s(u(x,t)) = \begin{pmatrix} 0 \\ \Delta p_{fr} \\ \rho \dot{q}_e \end{pmatrix} - \frac{1}{A} \frac{dA}{dt} \begin{pmatrix} \rho \\ \rho v \\ \rho u_0 + p \end{pmatrix} \quad (4)$$

3 Conservative methods

An approach to discretize the governing equations of thermo-fluid dynamics is now introduced based on Toro [34]. It is formulated in conserved variables and therefore called a conservative method.

The use of conservative methods is motivated by the presence of discontinuities such as shock waves in the solution of certain problems such as gas dynamics. Hou and LeFloch [13] have shown that formulations based on variables other than the conserved ones fail to correctly predict the solution at shock waves. They result in wrong jump conditions and thus wrong shock strength, speed, and location. The theorem of Lax and Wendroff [17] in turn states that conservative methods, if convergent, do converge to the weak solution of the conservation law. Consequently, conservative methods are an obvious choice if shock waves are potentially contained in the solution.

In this section, the compact formulation of the conservation laws introduced in equation (1) is used. The vector of conserved quantities is denoted by $u(x,t) = (\rho, \rho v, \rho u_0)$. In order to include weak solutions of (1), an integral form of the equations is used, a finite volume method.

As done in several numerical methods, the problem domain is discretized on a suitable computational mesh. The control volumes are defined based on a grid of cell side coordinates on an interval $[a, b]$

$$a = x_{1/2} < x_{3/2} < \dots < x_{n-1/2} < x_{n+1/2} = b \quad (5)$$

Based on it, cells, cell centers and cell sizes are defined for $i = 1, 2, \dots, n$.

$$\begin{aligned} I_i &= [x_{i-1/2}, x_{i+1/2}] \\ x_i &= \frac{1}{2}(x_{i-1/2} + x_{i+1/2}) \\ \Delta x_i &= x_{i+1/2} - x_{i-1/2} \end{aligned} \quad (6)$$

In this notation, $x_{i+1/2}$ is the coordinate of the right side of a computational cell I_i with cell center x_i . This grid is colocated. Furthermore, the maximum cell size is defined as follows.

$$\Delta x = \max_{1 \leq i \leq n} (\Delta x_i) \quad (7)$$

The discretization scheme allows to deduce algebraic equations or differential algebraic equations that properly approximate the governing equations. Note that, in the context of Modelica, the goal is to deduce differential algebraic equations and thus the equations have only to be discretized in space, not in time (“semi-discretized”).

The set of cell centers, which is used in a discretization scheme to deduce such equations for each cell, is called the stencil. For the most simple schemes, the stencil for cell I_i includes I_i itself and the cells to the left and to the right,

$$S(i) = \{I_{i-1}, I_i, I_{i+1}\} \quad (8)$$

Therefore, equation (1) is integrated over the interval I_i to obtain

$$\frac{d\bar{u}(x_i, t)}{dt} = s(\bar{u}(x_i, t)) - \frac{1}{\Delta x_i} (f(u(x_{i+1/2}, t)) - f(u(x_{i-1/2}, t))) \quad (9)$$

Herein, a cell average is used

$$\bar{u}(x_i, t) = \frac{1}{\Delta x_i} \int_{x_{i-1/2}}^{x_{i+1/2}} u(\xi, t) d\xi$$

Equation (9) is approximated by a semi-discretized conservative scheme, which results in a differential algebraic equation,

$$\frac{d\bar{u}_i(t)}{dt} = s(\bar{u}_i(t)) - \frac{1}{\Delta x_i} (f_{i+1/2} - f_{i-1/2}) \quad (10)$$

Herein, $\bar{u}_i(t)$ is a numerical approximation of the exact cell average $\bar{u}(x_i, t)$, and $f_{i\pm 1/2}$ is a numerical flux, an approximation of the physical flux $f(u(x_{i\pm 1/2}, t))$.

The remainder of this section is concerned with the construction of numerical fluxes. All these fluxes consist of a monotone flux and a reconstruction. Practically, a monotone flux is a flux free of spurious oscillations. Due to Godunov’s Theorem such linear fluxes are however first-order accurate only. Therefore, these monotone fluxes are often used together with reconstructions in order to build higher-order schemes. The reconstruction provides an approximation of the vector of conserved variables u (or any other variable of interest) based on the cell averages. Its higher-order accuracy yields, together with a first-order monotone flux, higher-order numerical flux.

3.1 Monotone flux and first-order schemes

A monotone numerical flux is defined using a function g ,

$$f_{i+1/2} = g(u_{i+1/2}^-, u_{i+1/2}^+) \quad (11)$$

Here, $u_{i+1/2}^-$ is in general an approximation of the vector of conserved variables at $x_{i+1/2}$ in the left limit, and $u_{i+1/2}^+$ in the right limit. Each monotone flux can be used without reconstruction with the approximation $u_{i+1/2}^- \approx u_i$ and $u_{i+1/2}^+ \approx u_{i+1}$. The results are first-order schemes. Alternatively, any more sophisticated approach may be used to reconstruct $u_{i+1/2}^\pm$.

In the following presentation of monotone fluxes, q_r will refer to the right limit $q_{i+1/2}^+$ of a quantity q . Similarly, q_l is abbreviated as $q_{i+1/2}^-$.

Monotone fluxes are classified as either upwind methods or central methods. Upwind methods discretize equations on a mesh according to the direction of propagation of information on that mesh. Central methods do not make a distinction based on the direction of information propagation. Within the upwind methods, both Godunov-type methods and flux vector splitting methods are presented based on [34].

3.1.1 Godunov-type Upwind Methods

These methods are also called flux difference splitting methods or Riemann approach methods. In the general case, $u_{i+1/2}^- \neq u_{i+1/2}^+$, i.e., at position $x_{i+1/2}$ a discontinuity is present. The original Godunov monotone flux therefore interpreted this as Riemann problem and provided the conserved variables at $x_{i+1/2}$, $u_{i+1/2}$. This is the state that will be present instantly at this position and will remain constant over a time step. Then, the flux can be evaluated at this position, $f(u_{i+1/2})$. The result is the Godunov monotone flux.

As the Godunov monotone flux uses the exact solution to the Riemann problem, the resulting method is computationally relatively expensive and is rarely used for practical computations. Godunov-type monotone fluxes follow the approach of the Godunov monotone flux but employ an approximate Riemann solver. This reduces the computational expense significantly and results in rather accurate monotone fluxes.

Roe’s Monotone Flux: This Godunov-type flux uses one of the most well-known approximate Riemann solvers. The approximate Riemann solver is due to Roe [26] and works as follows. The original Riemann problem is replaced by an approximate Riemann problem, which is solved exactly. The approximate problem is based on linearized conservation laws, $u_t + A_{I_r} u_x = 0$.

The linearized problem has to be established for each combination of governing equations (e.g., Euler equations) and thermodynamic property model (e.g., ideal gas).

Roe [26] established a methodology using averaged values such that $A_{lr} \left(u_{i+1/2}^+ - u_{i+1/2}^- \right) = A_{lr}(\bar{u})$ fulfills the given conditions. The vector \bar{u} is the vector of Roe average values. For the one-dimensional Euler equations and ideal gas, the Roe average values are as follows.

$$\begin{aligned}\bar{\rho} &= \frac{\rho_r + \rho_l}{\sqrt{\rho_r} + \sqrt{\rho_l}} \\ \bar{v} &= \frac{\sqrt{\rho_r} v_r + \sqrt{\rho_l} v_l}{\sqrt{\rho_r} + \sqrt{\rho_l}} \\ \bar{h}_0 &= \frac{\sqrt{\rho_r} h_{0,r} + \sqrt{\rho_l} h_{0,l}}{\sqrt{\rho_r} + \sqrt{\rho_l}}\end{aligned}$$

and

$$\bar{c}^2 = (\kappa - 1) \left(\bar{h}_0 - \frac{1}{2} \bar{v}^2 \right)$$

Due to specific properties [26], the linearized system can be transformed into a system of independent transport equations. The data difference $\Delta u = u_r - u_l$ is projected onto the right eigenvectors of A_{lr} . This establishes the wave strengths α_i . Proper integral relations allow to establish the numerical flux as

$$g_{Roe}(u_l, u_r) = \frac{1}{2} (f_l + f_r) - \frac{1}{2} \sum_{i=1}^3 \alpha_i |\lambda_i| K_i$$

with eigenvalues λ_i and right eigenvectors K_i .

For the problem of interest, the wave strengths are

$$\begin{aligned}\alpha_1 &= \frac{1}{2\bar{c}} [\Delta m - \Delta \rho (\bar{v} + \bar{c})] - \frac{1}{2} \alpha_2 \\ \alpha_2 &= -\frac{\kappa - 1}{\bar{c}^2} [\Delta \rho (\bar{v}^2 - h) - \bar{v} \Delta m + \Delta \bar{e}] \\ \alpha_3 &= \Delta \rho - \alpha_1 - \alpha_2\end{aligned}$$

Here, the data difference Δm for example refers to the difference in momentum.

HLL Monotone Flux: The Harten, Lax and van Leer [12] monotone flux simplifies the approximate Riemann problem even further. It neglects the contact surfaces and consequently assumes that between the shock and the expansion fan only a single homogeneous state is present. For hyperbolic systems of two equations this is correct, but for the Euler equations addressed herein this is a rough approximation. Even if the resolution of contact surfaces is poor, this monotone flux is still a robust and efficient one, whose accuracy is, on global level, often sufficient.

An advantage of this flux is that it can be applied easily to different thermodynamic property models. The approximate Riemann solver of Roe for example is not straight-forward to apply to several problems

such as ones involving real gas equations. It is therefore a relevant candidate for equation-based, object-oriented modeling languages applications, as the specific thermodynamic property models are often factored out of the component models, in which the discretized Euler equations are implemented.

The scheme is implemented via an a-priori estimation for the fastest signal speeds and its monotone flux is defined as

$$\begin{aligned}g_{HLL}(u_l, u_r) &= \frac{c_r^+ f(u_l) - c_l^- f(u_r)}{c_r^+ - c_l^-} \\ &+ \frac{c_r^+ c_l^-}{c_r^+ - c_l^-} (u_r - u_l)\end{aligned}$$

Here, the signal speeds are $c_r^+ = \max(0, v_r + c_r, \bar{v} + \bar{c})$ and $c_l^- = \min(0, v_l - c_l, \bar{v} - \bar{c})$ respectively. In these equations the Roe average velocity \bar{v} and the Roe average speed of sound \bar{c} have been used.

3.1.2 Flux Vector Splitting Upwind Methods

In Patankar [19] for instance, a simple first-order upwind scheme in primitive variables was introduced. Based on the sign of a characteristic quantity (usually, this is a velocity normal to the cell boundary), any variable on the boundary was established to have either the value from the left or the right side. In the context of the present approach to conservative methods and high-speed compressible flow, there is no simple scheme of this type. This becomes obvious from the hyperbolicity of the Jacobian $\partial f / \partial u$ and its eigenvalues.

In general, the real part of the eigenvalues can have any sign and a simple one-sided differencing scheme will be appropriate only if the real parts of all eigenvalues have the same sign. The general system will however have some eigenvalues with a positive real part, and one side will be upwind for them, while the others have a negative sign on the real part and consequently the upwind side will be opposite for them. A typical way to resolve this problem is to split such a system into one with a positive real part of the eigenvalues and one with a negative real part and to treat them separately. These are the flux vector splitting methods discussed in this section.

The flux vector splitting approach is also called Boltzmann approach and works as follows [34]. As before, the Jacobian of the system of nonlinear hyperbolic conservation laws (1) is of interest.

$$A(u) = \frac{\partial f(u)}{\partial u}$$

Due to hyperbolicity, it may be expressed as

$$A = K\Lambda K^{-1} \quad (12)$$

Here, Λ is the diagonal matrix of eigenvalues λ_i of A . The matrix K is the matrix of right column eigenvectors K_i . The flux vector splitting methods aim at splitting the flux $f(u)$ into components $f^+(u)$ and $f^-(u)$ based on the following equality.

$$f(u) = f^+(u) + f^-(u)$$

Following the introduction of this section, the split fluxes are established such that the eigenvalues $\hat{\lambda}_i^+$, $\hat{\lambda}_i^-$ of the Jacobian

$$\hat{A}^+ = \frac{\partial f^+(u)}{\partial u},$$

$$\hat{A}^- = \frac{\partial f^-(u)}{\partial u}$$

fulfill $Re(\hat{\lambda}_i^+) \geq 0$ and $Re(\hat{\lambda}_i^-) \leq 0$.

The Steger-Warming Monotone Flux: In order to establish such a splitting, the homogeneity property of (1) may be exploited. If the system of hyperbolic conservation laws fulfills this property, then

$$f(u) = A(u)u \quad (13)$$

like in the linear constant coefficient case. The unsteady Euler equations fulfill this property and consequently the splitting may utilize the structure exposed in (12), that is, the splitting may be applied to the diagonal matrix Λ . Steger and Warming [30] proposed a splitting of the eigenvalues λ_i ,

$$\lambda_i = \lambda_i^+ + \lambda_i^- \quad (14)$$

Here, $\lambda_i^+ \geq 0$ and $\lambda_i^- \leq 0$. Consequently, Λ is split as

$$\Lambda = \Lambda^+ + \Lambda^- \quad (15)$$

Λ^\pm are the diagonal matrices of the split eigenvalues λ_i^\pm . This leads directly to the splitting of A .

$$A = A^+ + A^- \quad (16)$$

where $A^\pm = K\Lambda^\pm K^{-1}$. If the property (13) is fulfilled, one arrives at an expression for the flux splitting.

$$f(u) = f^+(u) + f^-(u) \quad (17)$$

Here, $f^\pm(u) = A^\pm u$.

The crucial question is how to choose λ_i^\pm in (14). Steger and Warming [30] suggested to use to following equations.

$$\lambda_i^+ = \frac{1}{2}(\lambda_i + |\lambda_i|) = \max(\lambda_i, 0) \quad (18)$$

$$\lambda_i^- = \frac{1}{2}(\lambda_i - |\lambda_i|) = \min(\lambda_i, 0) \quad (19)$$

When exercising this approach, the following Steger-Warming monotone flux is established.

$$g_{sw}(u) = f^+(u) + f^-(u)$$

with

$$f^\pm(u) = \frac{\rho}{2\kappa} \begin{bmatrix} \lambda_1^\pm + 2(\kappa - 1)\lambda_2^\pm + \lambda_3^\pm \\ (v - c)\lambda_1^\pm + 2(\kappa - 1)v\lambda_2^\pm + (v + c)\lambda_3^\pm \\ (h - vc)\lambda_1^\pm + (\kappa - 1)v^2\lambda_2^\pm + (h + vc)\lambda_3^\pm \end{bmatrix}$$

The eigenvalues are given by (18) and (19). The remaining variables have to be evaluated according to the definition of the flux, i.e., for $f^+(u)$ the values from the left such as ρ_l , u_l and for $f^-(u)$ the values from the right such as ρ_r , u_r .

3.1.3 Centered Methods

Schemes, whose support does not depend on the sign of the characteristic speeds, are called centered schemes.

The Rusanov Monotone Flux, a local Lax-Friedrichs Flux: The Lax-Friedrichs flux is one of the simplest and most approximate methods considered herein. It was originally developed in the context of finite-difference methods and later applied to the finite-volume context.

Similarly to the HLLC method, only an expansion and a compression wave are considered. In the original Lax-Friedrichs flux, the speed of each wave was assumed to be such that it reached the cell boundaries exactly within a time step Δt . For uniform grids, each wave of the global problem therefore had the same speed, which is an even more approximate solution than in the HLLC method. As, in the present context, no fully explicit scheme is employed but the method of lines, no time step Δt is defined. For this reason and to slightly improve accuracy, a local form of the Lax-Friedrichs monotone flux, the Rusanov monotone flux [27], is considered. In the Lax-Friedrichs flux,

$$g_{LF}(u_l, u_r) = \frac{1}{2}(f(u_r) + f(u_l)) - \frac{1}{2} \frac{\Delta x}{\Delta t} (u_r - u_l)$$

the signal speed $\Delta x/\Delta t$ is replaced by $\lambda_{max} = \max((|v| + c)_l, (|v| + c)_r)$. Then, the Rusanov monotone flux is defined as follows.

$$g_{Rus}(u_l, u_r) = \frac{1}{2}(f(u_r) + f(u_l)) - \frac{1}{2}\lambda_{max}(u_r - u_l)$$

First-Order Centered Monotone Flux: The First-Order Centered Monotone flux (FORCE scheme) [33] is obtained when replacing the random sampling of Riemann problems in Random Choice Methods with deterministic integral averages.

According to Toro [34], for fully explicit schemes, the result is the arithmetic mean of the Lax-Friedrichs and Richtmyer [25] fluxes. The Richtmyer flux is a second-order scheme with constant coefficients and is thus, according to Godunov's classic theorem [9], not monotone and results in spurious oscillations.

For the fully explicit version of the Richtmyer flux, an intermediate state is first defined,

$$u_{Ri} = \frac{1}{2}(u_l + u_r) + \frac{1}{2} \frac{\Delta t}{\Delta x} (f(u_l) + f(u_r))$$

and then the flux is evaluated at it.

$$g_{Ri}(u_l, u_r) = f(u_{Ri})$$

Then, the FORCE flux is the arithmetic mean of the Lax-Friedrichs and Richtmyer fluxes [34]

$$g_{Force}(u_l, u_r) = \frac{1}{2}(g_{LF}(u_l, u_r) + g_{Ri}(u_l, u_r))$$

Again, the local version of the Lax-Friedrichs flux (the Rusanov flux presented in previous section) and a local version of the Richtmyer flux are used, is again obtained by replacing $\Delta x/\Delta t$ with λ_{max} .

After introducing some monotone numerical fluxes, methods to obtain higher-order approximations of the solution to (1) are considered.

3.2 Total Variation Diminishing schemes

Godunov's theorem [9] was mentioned already. It provides the theoretical foundation to the observation that *linear* second-order schemes are more accurate in smooth regions of a problem solution to (1) than first-order schemes. Near strong gradients and shocks, these methods produce spurious oscillations however. Monotone methods however do not exhibit such spurious oscillations. In case of linear schemes, their limited first-order accuracy is disadvantageous however.

One option to eliminate or reduce spurious oscillations for higher-order methods is to introduce artificial viscosity. This can be tuned such that it is large

enough to suppress oscillations in the neighborhood of discontinuities and small elsewhere to maintain accuracy. A disadvantage of this approach is however, that the quantity of artificial viscosity is problem dependent and therefore requires fine-tuning by the user. This approach is not followed here and instead a less empirical approach to introduce viscosity is adopted.

Therefore, in order to circumvent the limitations formulated by Godunov's theorem, schemes with variable coefficients, i.e., nonlinear schemes, are considered. Such schemes can adapt themselves to the local nature of the solution.

Harten [10] defined High-Resolution Methods as numerical methods with the following properties.

1. Second or higher-order of accuracy in smooth parts of the solution
2. The solution is free of spurious oscillations.
3. The resolution of discontinuities in the solution is high, i.e., the number of cells containing the numerical reproduction of the discontinuity is smaller in comparison with that of first-order monotone schemes.

A class of methods fulfilling these properties is that of Total Variation Diminishing methods [10]. See this reference for a definition of the total variation. For brevity, only the case of a smooth function $u(t)$, for which the total variation is

$$TV(u) = \int_{-\infty}^{\infty} |u'(x)| dx$$

and the case of a mesh function $u^n = \{u_i^n\}$ are mentioned. For the latter, the total variation is defined as

$$TV(u^n) = \sum_{i=-\infty}^{\infty} |u_{i+1}^n - u_i^n|$$

Fundamental properties of the exact solution of the conservation law (1) such as no creation of new local extrema lead to the conclusion that the total variation $TV(u(t))$ is a decreasing function of time [10]. Consequently, Total Variation Diminishing methods mimic a property of the exact solution.

For a general scalar conservation law, Harten [10] provided a theorem on a sufficient condition for a particular class of nonlinear schemes with two coefficients to be Total Variation Diminishing (TVD). These conditions are essentially four inequalities on these two coefficients. As the coefficients may in general be data dependent, Harten's theorem provides a tool for

the construction of nonlinear schemes that circumvent Godunov's theorem stated above.

The classic TVD approach to adaptively switch between the characteristics of a monotone first-order numerical flux g^{LO} and those of a higher-order constant coefficient flux g^{HI} is to make the following assumption [32].

$$g^{TVD} = g^{LO} + \varphi [g^{HI} - g^{LO}]$$

Here, φ is a flux limiter function that implements the adaptive algorithm. Analysis of Harten's theorem led to the identification of the Sweby TVD region [32]. In this region, various flux limiters have been defined such as the well-known limiters Superbee, Minbee, and Ultrabee.

In the following sections, this approach is not followed directly. Instead of flux limiters, slope limiters are used, which are analogous to the flux limiters.

For the reasons described in section 3.1.3, both an upwind TVD and a central TVD method are considered.

3.2.1 A MUSCL Upstream TVD Scheme

Van Leer [37, 38, 39] introduced a higher-order method along the concept of reconstruction mentioned in the introduction of this paper. MUSCL stands for Monotone Upstream-Centered Scheme for Conservation Laws.

The first-order schemes discussed so far use monotone fluxes directly by assuming piecewise constant data over the cells I_i , i.e., $u_{i+1/2}^- \approx u_i$ and $u_{i+1/2}^+ \approx u_{i+1}$. In the simplest MUSCL scheme, piecewise linear local reconstructions are used. The reconstruction has to maintain the integral average, which is trivially fulfilled for piecewise linear local reconstructions.

First, slope vectors $\Delta_{i\pm 1/2}$ are defined as follows.

$$\Delta_{i-1/2} = u_i - u_{i-1} \quad (20)$$

$$\Delta_{i+1/2} = u_{i+1} - u_i \quad (21)$$

Strictly speaking, these slopes are not slopes but differences of the vector of conserved quantities in adjacent cells. The terminology used in literature is adopted however and therefore $\Delta_{i\pm 1/2}$ are called slope vectors. In order to implement a TVD scheme, the approach of limited slopes described by Quirk [23] is used.

$$\hat{\Delta}_i = \begin{cases} \max[0, \min(\beta\Delta_{i-1/2}, \Delta_{i+1/2}), \min(\Delta_{i-1/2}, \beta\Delta_{i+1/2})] & \Delta_{i+1/2} > 0 \\ \min[0, \max(\beta\Delta_{i-1/2}, \Delta_{i+1/2}), \max(\Delta_{i-1/2}, \beta\Delta_{i+1/2})] & \Delta_{i+1/2} < 0 \end{cases}$$

The value $\beta = 1$ does, in the scalar case, reproduce the Minbee flux limiter, and $\beta = 2$ the Superbee flux limiter.

Based on the piecewise linear local reconstruction,

$$u_i(x, t) = \bar{u}_i(t) + \frac{x - x_i}{\Delta x_i} \hat{\Delta}_i$$

The values at the extreme points of the cell I_i are established.

$$u_{i-1/2}^+ = \bar{u}_i - \frac{1}{2} \hat{\Delta}_i \quad (22)$$

$$u_{i+1/2}^- = \bar{u}_i + \frac{1}{2} \hat{\Delta}_i \quad (23)$$

In order to finally obtain the second-order accurate upstream flux, some first-order monotone upstream flux is employed with the reconstructed values $u_{i+1/2}^-$, $u_{i+1/2}^+$.

$$g_{i+1/2}^{TVDu} = g_{i+1/2}^{mu} (u_{i+1/2}^-, u_{i+1/2}^+)$$

Note that $u_{i+1/2}^-$ is obtained from a reconstruction in cell I_i , and $u_{i+1/2}^+$ from a reconstruction in cell I_{i+1} .

3.2.2 A MUSCL Centered TVD Scheme

As mentioned before, also a second-order TVD centered scheme is introduced. It also follows the concept of the MUSCL scheme but uses a first-order monotone *centered* flux.

This approach is based on a slope limiter ξ_i , for which the following equation holds.

$$\hat{\Delta}_i = \xi_i \Delta_i$$

Here, the slope vector of the cells, Δ_i , is used.

$$\Delta_i = \frac{1}{2} (1 + \omega) \Delta_{i-1/2} + \frac{1}{2} (1 - \omega) \Delta_{i+1/2}$$

This is a weighted average of the side slope vectors $\Delta_{i\pm 1/2}$, see (20) and (21). The weighting parameter has to fulfill $\omega \in [-1, 1]$. In computations conducted for this paper, the value of $\omega = 0$ was used. Additionally, the ratio r_i of the cell side slope vectors is introduced.

$$r_i = \frac{\Delta_{i-1/2}}{\Delta_{i+1/2}}$$

Then, a slope limiter analogous to the Superbee flux limiter is [34]

$$\xi_{sb}(r) = \begin{cases} 0 & r \leq 0 \\ 2r & 0 \leq r \leq \frac{1}{2} \\ 1 & \frac{1}{2} \leq r \leq 1 \\ \min(r, \xi_r(r), 2) & r \geq 1 \end{cases}$$

A van Leer-type slope limiter is [34]

$$\xi_{vl}(r) = \begin{cases} 0 & r \leq 0 \\ \min\left(\frac{2r}{1+r}, \xi_r(r)\right) & r \geq 0 \end{cases}$$

A Minbee-type slope limiter is [34]

$$\xi_{mb}(r) = \begin{cases} 0 & r \leq 0 \\ r & 0 \leq r \leq 1 \\ \min(1, \xi_r(r)) & r \geq 1 \end{cases}$$

Above, $\xi_r(r)$, a TVD region limit that is defined as follows, was used.

$$\xi_r(r) = \frac{2}{1 - \omega + (1 + \omega)r}$$

As before, the conservative variable vector is approximated via the limited slope $\hat{\Delta}_i$ and equations (22) and (23). Then, the second-order accurate centered flux is obtained via a first-order monotone centered flux with the reconstructed values $u_{i+1/2}^-, u_{i+1/2}^+$. For this purpose, the FORCE flux can be used.

$$g_{i+1/2}^{TVDC} = g_{i+1/2}^{Force} \left(u_{i+1/2}^-, u_{i+1/2}^+ \right)$$

Note again that $u_{i+1/2}^-$ is obtained from a reconstruction in cell I_i , and $u_{i+1/2}^+$ from a reconstruction in cell I_{i+1} .

3.3 Weighted Essentially Non-Oscillatory schemes

One disadvantage of TVD schemes is that the accuracy near discontinuities is reduced. In the schemes presented above, this was directly visible in the slope for example. Also, the accuracy necessarily is reduced to first-order near *smooth* extrema.

In this section, both Essentially Non-Oscillatory and Weighted Essentially Non-Oscillatory schemes are presented, which are self-similar (i.e., there is no mesh size dependent parameter), uniformly high-order accurate, yet essentially non-oscillatory for piecewise smooth functions (i.e., the magnitude of the oscillations decays with order of accuracy of the scheme). Piecewise smooth functions are smooth except at finitely many isolated points. At these points, the function and its derivatives are assumed to have finite left and right limits.

The key element of these schemes is the reconstruction. This is a specific interpolation technique, which was developed for piecewise smooth functions. It works by automatically choosing the locally smoothest

stencil, and by that avoiding crossing discontinuities in the interpolation procedure as much as possible.

The Essentially Non-Oscillatory reconstruction algorithm starts with a stencil containing one or two cells only. It then adds either the cell to the right or the one to the left of the stencil, depending on which results in the less oscillatory interpolant.

Instead of choosing one of the candidate stencils and discarding the others, Weighted Essentially Non-Oscillatory reconstruction uses a convex combination of the interpolant through all candidate stencils.

First, the given two reconstructions are presented and then it is described how to establish a numerical flux from the corresponding reconstructions. This section is based on Shu [28].

3.3.1 Essentially Non-Oscillatory Reconstruction

Before describing the Essentially Non-Oscillatory (ENO) reconstruction, an important detail of interpolation methods used for reconstruction has to be addressed. In section 3.2 it was mentioned that linear interpolation in the MUSCL scheme was uncritical with respect to maintaining the proper cell average of the interpolant. In the context of the present methods, higher-order interpolation is considered and therefore the interpolant must be established in a way that maintains the cell average.

Assume that some function, say, velocity, is considered. The cell averages \bar{v}_i of that function $v(x)$ are given on a grid. One is interested in a polynomial $p_i(x)$ of degree $k - 1$ for each cell I_i . This then forms a k -th order approximation to $v(x)$ in the cell I_i . The polynomial shall be constructed such that its cell average shall agree with that of the original function \bar{v}_i .

Assume that, additionally to the cell I_i and the order of accuracy k , one is given a stencil $S(i)$ of k consecutive cells. The stencil is given via the left shift r , i.e., the stencil includes r cells to the left and s cells to the right of I_i , with $r + s + 1 = k$.

$$S(i) = \{I_{i-r}, \dots, I_{i+s}\} \tag{24}$$

In order to preserve the cell average, the interpolant over the stencil is established via the primitive function of $v(x)$.

$$V(x) = \int_{-\infty}^x v(\xi) d\xi$$

Then, the interpolant can be established. In computational implementations, this interpolation step is usually accelerated via the computation of so-called reconstruction coefficients. This is possible, because one

is usually not interested in the complete interpolant but only in values of it at specific stations such as $x_{i+1/2}$. Due to the linearity of the mapping from the cell averages \bar{v}_i to the interpolated values, these reconstruction coefficients depend on the left shift of the stencil r , the order k , and the mesh spacing Δx_i , but not on the function v itself.

The actual ENO approximation is addressed next. Here, an adaptive stencil is used. This means that the left shift r is not constant. A left shift r that is constant over the cells I_i would lead to a fixed stencil approximation (e.g., a central stencil) for which it was shown that it leads to spurious oscillations if of order two or higher with constant coefficients. In ENO approximation, the left shift is thus established for each cell I_i in a way that avoids including a cell with a discontinuous change in the stencil.

Harten et al. [11] showed that a robust criterion to identify the stencil with the “smoother” interpolant is to choose the one with the smaller absolute value of the Newton divided difference.

Recall the definition of the Newton divided differences. For the primitive function $V(x)$ the 0-th degree divided difference is

$$V[x_{i-1/2}] = V(x_{i-1/2})$$

and the general j -th degree divided difference with $j \geq 1$ is defined as

$$V[x_{i-1/2}, \dots, x_{i+j-1/2}] = \frac{V[x_{i+1/2}, \dots, x_{i+j-1/2}] - V[x_{i-1/2}, \dots, x_{i+j-3/2}]}{x_{i+j-1/2} - x_{i-1/2}}$$

Similarly, the divided differences of the cell averages are

$$\bar{v}[x_i] = \bar{v}_i$$

and in general

$$\bar{v}[x_i, \dots, x_{i+j}] = \frac{\bar{v}[x_{i+1}, \dots, x_{i+j}] - \bar{v}[x_i, \dots, x_{i+j-1}]}{x_{i+j} - x_i} \quad (25)$$

Note that the zeroth degree divided difference of \bar{v}_i is identical to the first degree divided difference of $V(x)$ due to the definition of the primitive function.

$$V[x_{i-1/2}, x_{i+1/2}] = \frac{V(x_{i+1/2}) - V(x_{i-1/2})}{x_{i+1/2} - x_{i-1/2}} \quad (26)$$

$$= \bar{v}_i$$

This equality allows to express the divided differences of $V(x)$ of degree $j \geq 1$ by those of \bar{v}_i of degree $j \geq 0$.

Taking the derivative of the k -th degree interpolation polynomial $P(x)$ to approximate $V(x)$, one finds that only divided difference of \bar{v}_i of degree $j \geq 1$ are required to express $p(x)$.

The ENO approximation thus identifies the “smoothest” stencil in \bar{v}_i via a stencil of $V(x)$, which is labeled $\hat{S}(i)$. Notice that from the latter the corresponding stencil in \bar{v}_i can be identified via (26). First, the divided differences of the primitive function $V(x)$ are computed using (26) and, for degrees $j \geq 2$, using (25). Then, the algorithm starts with a two point stencil in $V(x)$,

$$\hat{S}_2(i) = \{x_{i-1/2}, x_{i+1/2}\}$$

This stencil is then consecutively enlarged for $l = 2, \dots, k$. From the preceding step the following stencil is known

$$\hat{S}_l(i) = \{x_{i+1/2}, \dots, x_{i+l-1/2}\}$$

and one of the neighboring points $x_{j-1/2}$ and $x_{j+l+1/2}$ is added to the stencil. If

$$|V[x_{j-1/2}, \dots, x_{j+l-1/2}]| < |V[x_{j+1/2}, \dots, x_{j+l+1/2}]|$$

then $x_{j-1/2}$ is added to $\hat{S}_l(i)$ to obtain $\hat{S}_{l+1}(i)$. If the inequality is not fulfilled, then $x_{j+l+1/2}$ is added to the stencil.

As soon as the stencil is completely established, Lagrange or Newton interpolation can be used to find the interpolants. In computational implementations the reconstruction coefficients mentioned at the beginning of this section are usually used instead. By the choice of the stencil the left shift r is established. Then, the proper reconstruction coefficients can be used to instantly establish the interpolated values at the interface locations.

Figure 1 illustrates the interpolants chosen by Essentially Non-Oscillatory schemes. For the example $\bar{v} = \{10, 10.4, 10.25, 10, 3, 2.5, 2.25, 2\}$ and $x = \{1, 2, 3, 4, 5, 6, 7, 8\}$ were assumed. First, consider the resulting interpolant for cell 3. The scheme described above starts the stencil with this cell and extends it twice (i.e., *order* – 1 times) to the left or right. As described, the schemes includes either neighbor point that results in a smoother interpolant according to the criterion of divided differences. For cell 3, the scheme once selects a cell to the left and once a cell to the right for inclusion in the stencil. For cell 4 in turn, including the right cell (cell 5) would lead to rather large gradients in the interpolant each time. Therefore, the stencil is extended twice to the left. The interpolant for cell 4

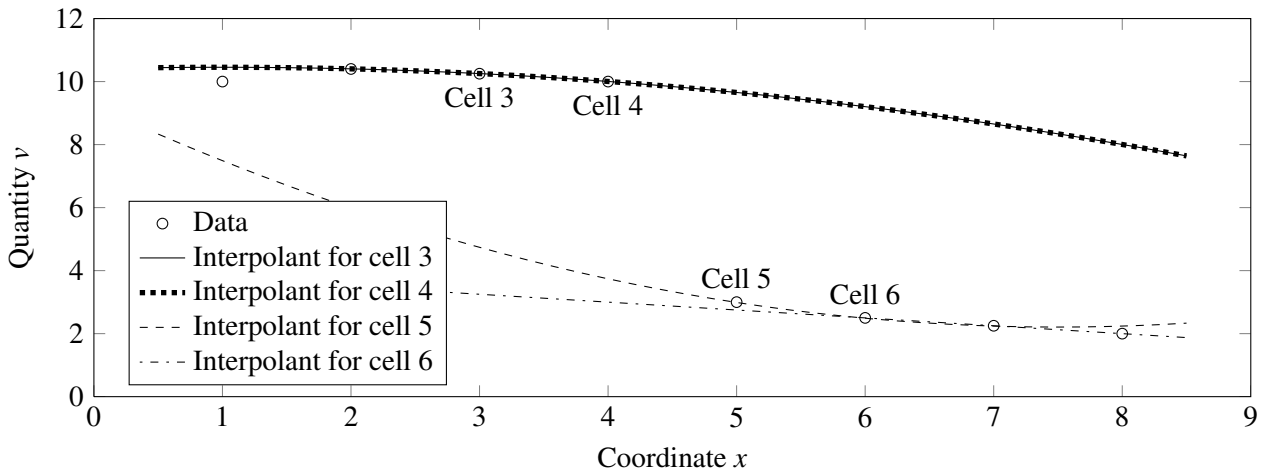


Figure 1: Third-order ENO reconstruction

is therefore identical to that of cell 3. For cells 5 and 6, the stencil is only extended to points to the right for similar reasons.

The left limit of $v_{4+1/2}$ is established based on the interpolant of cell 4, i.e., $v_{4+1/2}^- = 9.84$. The right limit is $v_{4+1/2}^+ = 3.33$.

3.3.2 Weighted Essentially Non-Oscillatory Reconstruction

ENO schemes are uniformly high-order accurate right up to the discontinuity, which is achieved by adaptively switching the stencil used for interpolation. However, certain properties leave room for improvements [28]:

- The stencil may change near zeros of the solution even by a round-off error perturbation.
- As the left shift of the stencil may change at neighboring points, the resulting numerical flux is not smooth.
- To the reconstruction scheme, $2k - 1$ cells are available. In the end, only k cells are used. This results in k -th order accuracy when $2k - 1$ -th order accuracy is theoretically possible in smooth regions of the solution.

The idea of Weighted Essentially Non-Oscillatory (WENO) reconstruction is to use a convex combination of the interpolants through several stencils. If, however, a candidate stencil contains a discontinuity, its weight shall be close to zero to mimic the successful properties of ENO schemes.

For each cell I_i k candidate stencils are consequently available.

$$S_r(i) = \{x_{i-r}, \dots, x_{i-r+k-1}\}$$

with $r = 0, \dots, k - 1$. Using the reconstruction coefficients, each stencil produces a different reconstruction of $v_{i+1/2}$, which is labeled $v_{i+1/2}^{(r)}$. A convex combination of these values is used to define the reconstruction using the WENO method.

$$v_{i+1/2} = \sum_{r=0}^{k-1} \omega_r v_{i+1/2}^{(r)}$$

For stability and consistency, $\omega_r \geq 0$ and $\sum_{r=0}^{k-1} \omega_r = 1$ need to be imposed. In smooth regions, these weights should approximate optimal high-order weights to $k - 1$ -th order, which would imply $(2k - 1)$ -th order of the complete reconstruction scheme. The question is now what these optimal weights are. In the general case, this leads to an overdetermined system of equations, which can be solved, e.g., by using a least-squares algorithm. In the case of a uniform mesh, the equation system becomes square and an explicit solution can be computed. Jiang and Shu [16] gave optimal weights d_r for uniform grids and $1 \leq k \leq 3$. Herein, $k = 3$ is considered. For this value of k , the following optimal weights have been established.

$$d_0 = \frac{3}{10}, \quad d_1 = \frac{3}{5}, \quad d_2 = \frac{1}{10}$$

Furthermore, Jiang and Shu [16] suggested the following form of the weights

$$\omega_r = \frac{\alpha_r}{\sum_{s=0}^{k-1} \alpha_s}$$

for $r = 0, \dots, k-1$. Coefficients α_r in turn are defined as follows

$$\alpha_r = \frac{d_r}{(\varepsilon + \beta_r)^2}$$

Here, $\varepsilon > 0$ is introduced to avoid division by zero. Following Jiang and Shu [16], $\varepsilon = 10^{-6}$ was used in computations. β_r are called smooth indicators in the given reference and have been defined as follows

$$\beta_r = \sum_{l=1}^{k-1} \int_{x_{i-1/2}}^{x_{i+1/2}} \Delta x^{2l-1} \left(\frac{\partial^l p_r(x)}{\partial x^l} \right)^2 dx$$

This is the sum of the squares of the scaled L^2 norms for all derivatives of the interpolation polynomial $p_r(x)$ over the interval $(x_{i-1/2}, x_{i+1/2})$. For $k = 3$, the result is a $2k - 1 = 5$ -th order accurate reconstruction.

Figure 2 illustrates Weighted Essentially Non-Oscillatory reconstruction on the same example as figure 1. The reconstruction of the left limit of $v_{4+1/2}$ is considered, i.e., $v_{4+1/2}^-$. For this, the scheme uses three stencils $S_r(4)$ with increasing left-shift r . The interpolants based on these stencils are illustrated in the figure. Note the strong gradients in the interpolants using $S_0(4)$ and $S_1(4)$. This is also an illustration that the stencil selection of the ENO scheme shown in figure 1 for cell 4 was reasonable.

The WENO scheme proceeds with the different reconstruction values $v_{4+1/2}^{(0)}$ to $v_{4+1/2}^{(2)}$, which are each marked with a filled circle in figure 2. For this particular example, the scheme results in weights $\omega_0 = 1.3 \cdot 10^{-6}$, $\omega_1 = 15.6 \cdot 10^{-6}$, $\omega_2 = 0.999983$. This means, that the interpolant with left-shift $r = 2$ dominates and $v_{4+1/2}^- \approx v_{4+1/2}^{(2)}$.

3.3.3 ENO and WENO numerical fluxes

So far, two different algorithms for the reconstruction of piecewise smooth functions were introduced. The question is now how to construct corresponding higher-order numerical fluxes for the system of hyperbolic conservation laws (1) from these reconstructions.

Probably, the easiest way to do this is to apply the reconstruction to each component of the vector of conserved variables u separately and thus reconstruct the left and right limit $u_{i+1/2}^\pm$ at the location $x_{i+1/2}$. Then, a monotone first-order flux can be used to establish an essentially non-oscillating higher-order numerical flux.

Shu [28] remarks that only low-order schemes are highly sensitive to the choice of first-order monotone

flux. This sensitivity decreases with increasing order of accuracy and therefore a simple Lax-Friedrichs monotone flux is used in the given reference to construct higher-order WENO numerical fluxes.

The given component-wise approach to construct a numerical flux based on ENO and WENO reconstructions is simple to implement. Also, the resulting schemes work reasonably well for many applications, in particular if the order of the scheme is not high. Shu [28] mentions “second or sometimes third-order”.

If the order of the scheme is high or a more demanding test problem shall be solved, the following characteristic decomposition is much more robust and should be implemented instead.

Recall the diagonal decomposition of the Jacobian of the flux in section 3.1.2 on flux vector splitting, (12). A change of variables $v = K^{-1}u$ leads to a decoupling of the system of conservation laws (1). Then, the component-wise application of the ENO or WENO reconstruction is fundamentally justified. The reconstructed values $v_{i+1/2}^\pm$ are then transformed back into the physical space of conserved variables,

$$u_{i+1/2}^\pm = K v_{i+1/2}^\pm$$

A remaining question is the choice of K , which depends on u , $K = K(u)$. For this purpose, the Roe averages introduced in section 3.1.1 were used, as this leads to advantageous properties such as the satisfaction of the mean value theorem.

Based on the reconstructed left and right limit $u_{i+1/2}^\pm$ at the location $x_{i+1/2}$, a monotone first-order flux is used again to establish an essentially non-oscillating higher-order numerical flux.

4 Object-oriented implementation

Two libraries for object-oriented modeling and simulation of gas dynamics were developed for [29] and this paper. Both were written in Modelica. The first one is a library specific to ideal gases, which allows several simplifications and results in little computational overhead. The second one is a gas dynamics library for generic thermodynamic property models. These thermodynamic property models are implemented according to the object-oriented interface MODELICA.MEDIA [6]. This interface had to be extended with two additional methods to be suitable for applications in gas dynamics. These and other implementation aspects are discussed in this section.

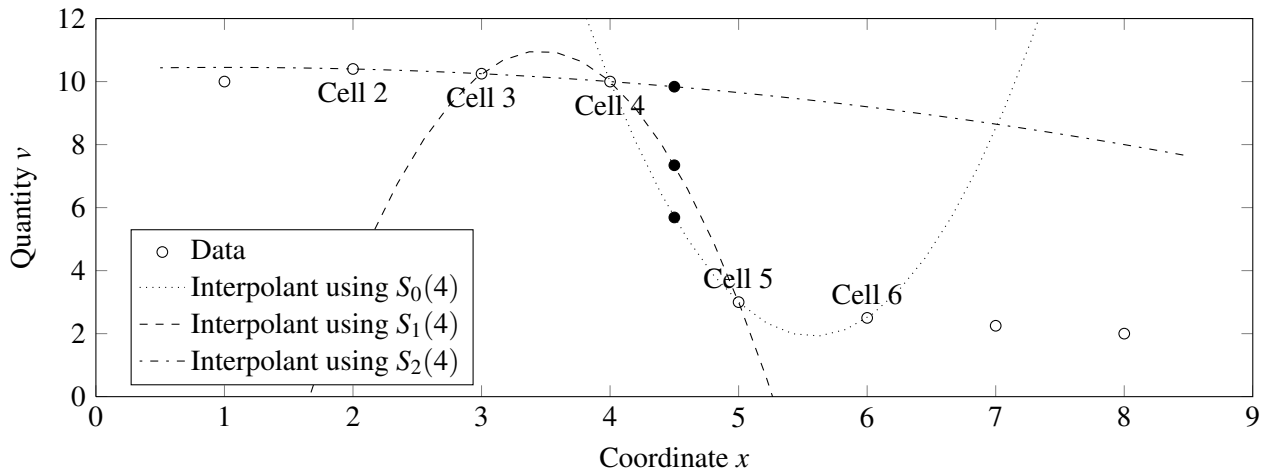


Figure 2: Fifth-order WENO reconstruction

4.1 Ideal gas and generic thermodynamic property models

A large fraction of the literature on discretization methods using conservative methods considers ideal gas equations of state only. Discretizations using real gas¹ equations of state in turn consider non-ideal media, too. Several articles make assumptions on the structure of the real gas equations of state however (e.g., Liou et al. [18] assume a “general pressure function” but require that it be explicit in density, specific internal energy, and mass fractions, and Gallouët et al. [8] explicitly assume Tammann and van der Waals equations of state).

In equation-based, object-oriented modeling and simulation, one aims to encapsulate the equations of state in separate classes and implement discretization methods independently using a generic interface. As the given real gas schemes require structural assumptions on the equations of state, too, a generic interface had to be extended with several methods specific to these structural assumptions. A clean separation between discretization scheme and equation of state appears to be difficult in this case.

A large fraction of the methods described in the previous section 3 are specific to ideal gases with constant specific heat capacity c_p . Specialized Riemann solvers can be constructed easily for some of these methods (such as the HLLC method described in section 3.1.1). In the context of equation-based, object-oriented modeling languages, such approximate Riemann solvers had to be exchanged *concurrently* with the equations of state. A more practical solution is the

¹In this thesis, a real gas is one that is not both thermally and calorically ideal.

use of centered schemes. These schemes are independent of any Riemann solver and can thus be used with any thermodynamic property model. As described in section 3.1.3, the support of these schemes does not depend on the sign of the characteristic speeds. While the upwind schemes as discussed in sections 3.1.1 and 3.1.2 are more accurate in several cases than their centered counterparts, they are usually more complex and computationally expensive [34]. Therefore, in the libraries described herein, monotone and TVD centered schemes as well as schemes using higher-order reconstruction with a centered scheme are implemented for general thermodynamic property models and upwind methods are restricted to ideal gases.

4.2 Generic interface to thermodynamic property computations

As described above, the object-oriented interface of MODELICA.MEDIA [6] is used for thermodynamic property computations. In order to be applicable to gas dynamics, this interface has to be extended with two additional methods.

The first extension is required for the conversion of conserved variables to primitive variables. In the gas dynamics library for generic equations of state the primitive variables are velocity v and the thermodynamic state record of the medium². For the conversion of the vector u as defined in equation (2) to the primitive variables an additional `setState` function is thus required. From u , density and specific internal energy can be established. Therefore, a function

²In place of the velocity the mass flow rate could have been used, too. This selection is ambiguous and was eventually made for similarity with conventional implementations of gas dynamics.

`setState_duX` is used.

The second extension is required for the conversion of the classic primitive variables $\{\rho, v, p\}$ to the ones used in the object-oriented implementation for generic thermodynamic property computations, the thermodynamic state record and velocity. This is necessary in case of a characteristic decomposition such as the one discussed in section 3.3.3. For this purpose, a function `setState_pdX` is required. Note that this is only required if a gas dynamics library for generic thermodynamic property models shall also be used with ideal gases.

4.3 Conservative and non-conservative formulations

In order to obtain valid simulation results, the conserved quantities in the governing equations and the conservation statements they imply have to make physical sense [34]. Formulations that are conservative purely in a mathematical sense (i.e., formally, they can be expressed as (1), but there is no corresponding conservation law in physics) will, in case of shock waves, result in wrong shock speeds and therefore wrong solutions [34].

In the context of equation-based, object-oriented modeling languages, a simple solution is to explicitly select the conserved variables themselves as state variables, i.e., $u(x, t)$. This is done in the gas dynamics library specific to ideal gases. For ideal gases that are both thermally and calorically ideal (in particular, c_p is not a function of temperature), all intensive quantities can be established in closed form based on any two thermodynamic potentials. Therefore, no distinction between independent and dependent variables is required for such media.

For generic thermodynamic property models this is different. In general, such models are explicit in a number of thermodynamic potentials only (e.g., pressure and specific enthalpy). As long as the physical flux is not changed, it is then possible to use the independent variables of a thermodynamic property model as state variables instead. This is the approach followed in the gas dynamics library for generic thermodynamic property models.

4.4 Inhomogeneous problems

In several references on computational methods for gas dynamics, fully explicit conservative methods are considered in contrast to (10). In the context of

equation-based, object-oriented modeling, it is natural however to use a semi-discretized formulation. Furthermore, this has advantages for inhomogeneous problems. No source term splitting schemes [31] are required for the present approach. With the semi-discretization (also called method of lines) both the numeric fluxes and the source term are algebraic expressions and no further complications arise for inhomogeneous problems.

4.5 Library design

In this section, the design of the two gas dynamics libraries is sketched. The one considering generic thermodynamic property models is emphasized and some remarks are made on the one specific to ideal gases. For readability, the code illustrates single-substance media only. Mass fractions of multiple-substance media can be covered analogously to the other primitive variables, because they are similarly dominated by convection.

The connector has to implement the stencil defined in equation (8). Its length depends on the stencil length required by the discretization scheme. If the stencil for a flux computation has to include n cells, then at least $n/2$ of these cells are inside the domain modeled by the respective component and need not be accessed via the connector. This implies that at most $n/2$ cells of the stencil have to be provided by the connector. Therefore, the connector definition given in listing 1 is used.

Note the replaceable discretization package (“Discretization”) in the connector definition in addition to the replaceable package containing the thermodynamic property model (“Medium”). A vector of thermodynamic states and one of velocities of the given length are defined twice. Different causal prefixes are used to handle how one component “prescribes” and “reads” which variables³. The library considering ideal gases only uses density and pressure vectors in place of the thermodynamic state.

Additionally, information about the computational mesh has to be included in the connector. In the proposed connector definition, the coordinates of the sides of the cells are used. They are defined in a local coordinate system, whose origin is set to the side shared by two components connected together. The coordinate of this shared side can thus be omitted and the same number of side coordinates and cell center variables on

³The causal prefixes are used in the acausal modeling language just to define a nominal causality, not an actual one.

```

1 connector Stencil_a
2   "Interface for quasi one-dimensional high-speed flow"
3
4   replaceable package Medium =
5     Modelica.Media.Interfaces.PartialMedium "Medium model";
6
7   replaceable package Discretization =
8     GasDynamics.Discretizations.Partial.PartialDiscretization
9     "Discretization";
10
11  output Medium.ThermodynamicState
12    state_a[Discretization.halfStencilLength]
13    "Thermodynamic state stencil";
14  output SI.Velocity v_a[Discretization.halfStencilLength]
15    "Velocity stencil";
16  output SI.Length x_side_a[Discretization.halfStencilLength]
17    "Cell side coordinate";
18
19  input Medium.ThermodynamicState
20    state_b[Discretization.halfStencilLength]
21    "Thermodynamic state stencil";
22  input SI.Velocity v_b[Discretization.halfStencilLength]
23    "Velocity stencil";
24  input SI.Length x_side_b[Discretization.halfStencilLength]
25    "Cell side coordinate";
26 end Stencil_a;

```

Listing 1: Connector for high-speed compressible flow

the thermodynamic state and velocity is included. The side coordinates for `Stencil_a` are defined strictly positive; those for `Stencil_b` strictly negative.

Analogous to the `Stencil_a` connector definition in listing 1, a connector `Stencil_b` is defined. It differs only in inverted causality prefixes (input instead of output and vice versa).

The discretization package contains structural parameters including the stencil length, conversion functions, an exchangeable thermodynamic properties model, and flux functions. Its interface is defined in listings 2 to 4.

The structural parameters of a `Discretization` are its name, whether it uses equations applicable to ideal gases, its order of accuracy, and the stencil length.

The conversion functions of a `Discretization` convert the set of primitive variables (thermodynamic state record and velocity) to the vector of conserved variables as defined in equation (2) and vice versa. Note that these functions need not be replaceable, because the implementations are generally valid. Note that in the second conversion function in listing 3 one of the

additional functions mentioned in section 4.2 is used (`setState_duX()`).

The key elements of a `Discretization` are the flux functions. Their interfaces are described in listing 4. For readability, interfaces are defined for both a monotone first-order flux and the arbitrary-order numerical flux. This allows to clearly separate the reconstruction and the Riemann solver for instance. In models, only the arbitrary-order numerical flux is used and therefore the use of the monotone flux function is optional. The monotone flux arguments are the left and right thermodynamic state and the flow velocities. It returns the flux vector. The arbitrary-order flux function has a stencil of thermodynamic states and of velocity as well as the cell side coordinates as inputs and also returns the flux vector. The `Discretization` package also contains a replaceable package implementing thermodynamic properties. This is not shown in listings 2 to 4. `Discretization` packages were implemented using the Local Lax-Friedrichs flux, Roe's Riemann solver, the HLLE Riemann solver, the Steger-Warming flux vector splitting, the First-Order Centered flux, the Muscl-

```

1 partial package PartialDiscretization
2   "Interface for discretization in compact flux form"
3
4   // Description
5   constant String discretizationName =
6     "unusablePartialDiscretization"
7     "Name of the discretization";
8
9   // Type of discretization
10  constant Boolean idealGasOnly = false
11    " = true, if contains specifics of ideal gases";
12  constant Integer order(min=1) = 1
13    "Order of discretization method";
14
15  // Stencil definition
16  constant Integer halfStencilLength = 1
17    "Half of length of stencil for flux f_(i+1/2)";
18  final constant Integer stencilLength = 2*halfStencilLength
19    "Length of stencil for flux f_(i+1/2)";
20
21  //...
22
23 end PartialDiscretization;

```

Listing 2: Discretization interface, structural parameters

Hancock TVD scheme with several limiters and monotone fluxes both in upstream and in centered versions, third- to ninth-order ENO schemes and several fifth-order WENO schemes with and without characteristic decomposition.

The implementation of a Discretization is illustrated for a second-order Muscl-Hancock scheme with a Superbee limiter and a Local Lax-Friedrichs flux in [29] and omitted here due to space constraints.

4.6 Applications

Results of a Sod-type problem are shown in figure 3. Here, the results of computations using the Local Lax-Friedrichs scheme (a first-order monotone centered method) are compared to those using a fifth-order WENO scheme (using Roe's first-order monotone flux and a characteristic decomposition). The figure illustrates the generally accepted result that proper higher-order reconstructions lead to higher resolution of shock waves, expansion fans, and contact discontinuities [34]. That is, such phenomena are smeared over fewer computational cells.

5 Conclusions

A conceptually meaningful structure for numerical gas dynamics using Modelica was introduced. The reviewed discretization schemes were implemented in the resulting framework and delivered robust and efficient simulation of the corresponding thermo-fluid dynamics problems.

References

- [1] W. Casas. *Untersuchung und Optimierung sorptionsgestützter Klimatisierungsprozesse*. PhD thesis, Technical University of Hamburg-Harburg, Institute of Thermo-Fluid Dynamics, 2006.
- [2] W. Casas and G. Schmitz. Experiences with a gas driven, desiccant assisted air conditioning system with geothermal energy for an office building. *Energ. Buildings.*, 37(5):493–501, 2005.
- [3] F. Casella and A. Leva. Modelica open library for power plant simulation: design and experimental validation. In P. Fritzson, editor, *Proceedings*

```

1 partial package PartialDiscretization
2   "Interface for discretization in compact flux form"
3
4   //...
5
6   function primitiveToConserved
7     "Convert primitive variables to conserved variables"
8     input Medium.ThermodynamicState state "Thermodynamic state";
9     input SI.Velocity v "Velocity";
10    output Real u[3] "Vector of conserved variables";
11  algorithm
12    u := {Medium.density(state), Medium.density(state)*v,
13         Medium.density(state)*
14         (Medium.specificInternalEnergy(state) + 1/2*v*v)};
15  end primitiveToConserved;
16
17  function conservedToPrimitive
18    "Convert conserved variables to primitive variables"
19    input Real u[3] "Vector of conserved variables";
20    output Medium.ThermodynamicState state "Thermodynamic state";
21    output SI.Velocity v "Velocity";
22  algorithm
23    v := u[2]/u[1];
24    state := Medium.setState_duX(u[1], u[3]/u[1]-1/2*v*v,
25                               Medium.X_default);
26  end conservedToPrimitive;
27
28  //...
29
30 end PartialDiscretization;

```

Listing 3: Discretization interface, conversion functions

- of the Third International Modelica Conference*, pages 41–50, Linköping, Sweden, 2003.
- [4] F. Casella and A. Leva. Modelling of thermo-hydraulic power generation processes using Modelica. *Math. Comput. Model. Dyn. Syst.*, 12(1):19–33, 2006.
- [5] J. Díaz López. Shock wave modeling for Modelica.Fluid library using oscillation-free logarithmic reconstruction. In *Proceedings of the Fifth International Modelica Conference*, pages 641–649, 2006.
- [6] H. Elmqvist, H. Tummescheit, and M. Otter. Object-oriented modeling of thermo-fluid systems. In P. Fritzson, editor, *Proceedings of the Third International Modelica Conference*, pages 269–286, Linköping, Sweden, 2003.
- [7] R. Franke, F. Casella, M. Otter, M. Sielemann, S.-E. Mattson, H. Olsson, and H. Elmqvist. Stream connectors—an extension of Modelica for device-oriented modeling of convective transport phenomena. In F. Casella, editor, *Proceedings of the seventh International Modelica conference*, pages 108–121, Como, September 2009.
- [8] T. Gallouët, J. Hérard, and N. Seguin. Some recent finite volume schemes to compute euler equations using real gas eos. *Int. J. Numer. Meth. Fl.*, 39(12):1073–1138, 2002.
- [9] S. K. Godunov. A finite difference method for the computation of discontinuous solutions of the equations of fluid dynamics. *Mat. Sbornik.*, 47:357–393, 1959.
- [10] A. Harten. High resolution schemes for hy-


```

1 partial package PartialDiscretization
2   "Interface for discretization in compact flux form"
3
4   //...
5
6   replaceable partial function monotoneFlux
7     "First-order flux approximation"
8     input Medium.ThermodynamicState state_l
9       "Stencil of thermodynamic states on left (i)";
10    input Medium.ThermodynamicState state_r
11      "Stencil of thermodynamic states on right (i+1)";
12    input SI.Velocity v_l "Velocity in x-dir on left, v_(i)";
13    input SI.Velocity v_r "Velocity in x-dir on right, v_(i+1)";
14    output Real flux[3] "Fluxes f_(i+1/2)";
15  end monotoneFlux;
16
17  replaceable partial function flux "Numeric flux approximation"
18    input Medium.ThermodynamicState state[stencilLength]
19      "Thermodynamic state stencil";
20    input SI.Velocity v[stencilLength] "Velocity stencil";
21    input Real x_side[stencilLength + 1]
22      "Coordinates of cell sides (i-1/2), (i+1/2) etc.";
23    output Real flux[3] "Fluxes f_(i+1/2)";
24  end flux;
25
26  //...
27
28 end PartialDiscretization;

```

Listing 4: Discretization interface, flux functions

- perbolic conservation laws. *J. Comput. Phys.*, 49:357–393, 1983.
- [11] A. Harten, B. Engquist, S. Osher, and S. Chakravarthy. Uniformly high order essentially non-oscillatory schemes, III. *J. Comput. Phys.*, 71:231–303, 1987.
- [12] A. Harten, P. D. Lax, and B. van Leer. On upstream differencing and Godunov-type schemes for hyperbolic conservation law. *SIAM Rev.*, 25(1):35–61, 1983.
- [13] T. Y. Hou and P. LeFloch. Why non-conservative schemes converge to the wrong solutions: Error analysis. *Math. Comput.*, 62:497–530, 1994.
- [14] J. Jensen, J. Jensen, and H. Tummescheit. Moving boundary models for dynamic simulations of two-phase flows. In *Proceedings of the Second International Modelica Conference*, 2002.
- [15] J. M. Jensen. *Dynamic Modeling of Thermo-Fluid Systems with focus on evaporators for refrigeration*. PhD thesis, Technical University of Denmark, Department of Mechanical Engineering, 2003.
- [16] G. Jiang and C.-W. Shu. Efficient implementation of weighted ENO schemes. *J. Comput. Phys.*, 126:202–228, 1996.
- [17] P. D. Lax and B. Wendroff. Systems of conservation laws. *Comm. Pure Appl. Math.*, 13:217–237, 1960.
- [18] M. Liou, B. Leer, and J. Shuen. Splitting of inviscid fluxes for real gases. *J. Comput. Phys.*, 87(1):1–24, 1990.
- [19] S. Patankar and D. Spalding. A calculation procedure for heat, mass and momentum transfer in three-dimensional parabolic flows. *Int. J. Heat. Mass. Tran.*, 15:1787–1806, 1972.

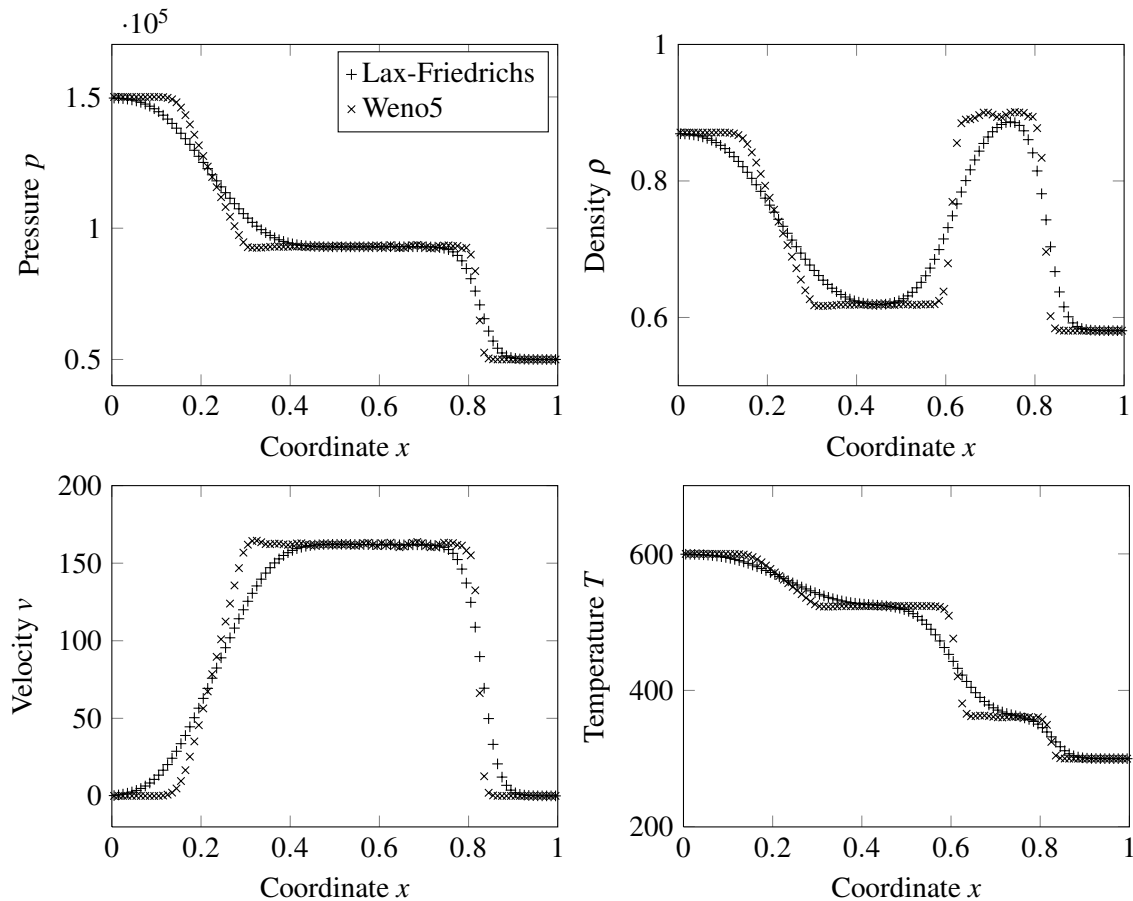


Figure 3: Comparison of Local Lax-Friedrichs and fifth-order WENO schemes on a Sod-type problem

- [20] T. Pfafferott. *Dynamische Simulation von CO₂-Kälteprozessen für mobile Anwendungen*. PhD thesis, Technical University of Hamburg-Harburg, Institute of Thermo-Fluid Dynamics, 2005.
- [21] K. Pröhl. *Untersuchung von Energie- und Massenspeicherungsvorgängen in Pkw-Kälteanlagen*. PhD thesis, Technical University of Hamburg-Harburg, Institute of Thermo-Fluid Dynamics, 2009.
- [22] K. Pröhl and G. Schmitz. Modeling of frost growth on heat exchanger surfaces. In *Proceedings of the Fifth International Modelica Conference*, 2006.
- [23] J. J. Quirk. An alternative to unstructured grids for computing gas dynamic flows around arbitrarily complex two dimensional bodies. *Comput. Fluid.*, 23(1):125–142, 1994.
- [24] C. C. Richter. *Proposal of New Object-Oriented Equation-Based Model Libraries for Thermodynamic Systems*. PhD thesis, Technical University Braunschweig, Institute for Thermodynamics, 2008.
- [25] R. D. Richtmyer and K. W. Morton. *Difference Methods for Initial Value Problems*. Interscience-Wiley, New York, 1967.
- [26] P. L. Roe. Approximate Riemann solvers, parameter vectors, and difference schemes. *J. Comput. Phys.*, 43:357–372, 1981.
- [27] V. V. Rusanov. Calculation of interaction of non-steady shock waves with obstacles. *USSR J. Comput. Math. Phys.*, 1:267–279, 1961.
- [28] C.-W. Shu. Essentially non-oscillatory and weighted essentially non-oscillatory schemes for hyperbolic conservation laws. *Advanced numerical approximation of nonlinear hyperbolic equations*, 1697:325–432, 1998.
- [29] M. Sielemann. *Device-Oriented Modeling and Simulation in Aircraft Energy Systems Design*. PhD thesis, Technical University of Hamburg-Harburg, Institute of Thermo-Fluid Dynamics, 2012.

- [30] J. L. Steger and R. F. Warming. Flux vector splitting of the inviscid gasdynamic equations with applications to finite difference methods. *J. Comput. Phys.*, 40:263–293, 1981.
- [31] G. Strang. On the construction and comparison of difference schemes. *SIAM J. Numer. Anal.*, 5(3):506–517, 1968.
- [32] P. K. Sweby. High resolution schemes using flux limiters for hyperbolic conservation laws. *SIAM J. Numer. Anal.*, 21:995–1011, 1984.
- [33] E. F. Toro. On two Glimm-related schemes for hyperbolic conservation laws. In *Proceedings of the Fifth Annual Conference of the CFD Society of Canada*, pages 3.49–3.54. University of Victoria, Canada, 1997.
- [34] E. F. Toro. *Riemann Solvers and Numerical Methods for Fluid Dynamics: A Practical Introduction*. Springer, 1997.
- [35] H. Tummescheit. *Design and Implementation of Object-Oriented Model Libraries using Modelica*. PhD thesis, Lund University, Department of Automatic Control, 2002.
- [36] H. Tummescheit, J. Eborn, and K. Prölb. Airconditioning—a Modelica library for dynamic simulation of AC systems. In G. Schmitz, editor, *Proceedings of the Fourth International Modelica Conference*, Hamburg, Germany, 2005.
- [37] B. van Leer. Towards the ultimate conservative difference scheme I. the quest for monotonicity. *Lect. Notes. Phys.*, 18:163–168, 1973.
- [38] B. van Leer. Towards the ultimate conservative difference scheme II. monotonicity and conservation combined in a second order scheme. *J. Comput. Phys.*, 14:361–370, 1974.
- [39] B. van Leer. Towards the ultimate conservative difference scheme III. upstream-centered finite difference schemes for ideal compressible flow. *J. Comput. Phys.*, 23:263–275, 1977.
- [40] J. Vasel and G. Schmitz. Transient simulation of a direct-evaporating CO₂ cooling system for an aircraft. In *25th International Congress of the Aeronautical sciences (ICAS), Proceedings of the*, Hamburg, Germany, September 2006.

Gas Exchange and Exhaust Condition Modeling of a Diesel Engine using the Engine Dynamics Library

Johan Dahl † Daniel Andersson ‡

†Volvo Group Truck Technology, Control Systems, Gothenburg, Sweden

‡Modelon AB, Lund, Sweden

Abstract

In this paper the newly developed Engine Dynamics Library is presented. Ever increasing consumer and regulatory demand for improved fuel economy and lower emissions forces the engines and Engine After-Treatment Systems (EATS) to be improved continuously. Since the complete system is very complex, models are useful in cost effectively developing new control strategies and select hardware. The library is based on a mean-value combustion model and the focus lies on modeling the gas exchange with real-time like simulation times, useful for engine optimization and for evaluation of control strategies. The library contains models of the standard engine components such as manifolds, pipe, turbines, compressors, valves, mechanics, etc. Simulation results from Dymola for a 13 L Volvo truck engine demonstrate that the model captures the transient flow and temperatures and emission trends, and has sufficient accuracy to be useful in engine optimization. The physical modeling approach allows for virtual prototyping by replacing individual components, which is an important advantage over black-box modeling. It is shown that the model captures essential system properties in the gas exchange, such as non-minimum phase behavior and sign reversal for VGT and EGR valve actuation. The model has been calibrated using surface fitting of maps and least-squares estimation of parameters in Matlab, as well as parameter optimization using JModelica and FMI.

Keywords: Engine modeling; Engine simulation; Air Gas management

1 Introduction

As the requirements on the engine and EATS become more strict, a new development process of control strategies and hardware concept selection is needed as only using engine test cells and vehicles in the development process is too time consuming and expensive.

In the new development process at Volvo, Software-In-the-Loop (SIL) simulations are used more extensively in the control strategy and hardware development. With the introduction of US10 and soon EU6 legislation ultra low on-road emissions are required. Future emission legislation will also include CO_2 , N_2O and NO_2 [1]. To fulfill these requirements with optimal fuel consumption, the significant interaction between the engine and EATS must be considered and control strategies for both components need to be optimized together [2]. This requires good engine models with accurate modeling of the engine out conditions. In particular, focus has been on predicting the sensitivity of the dynamic response and engine exhaust temperature with respect to the air gas management. Issues about control system design or strategy are not in the scope of this paper. Nevertheless, a good physical model of the engine provides useful insights for both the control system designers and hardware selection. The engine model is also useful for finding suitable requirements of the EATS system. For example the emission transient response can be a limiting requirement for the needed volumes of the Diesel Oxidation Catalyst (DOC), Diesel Particulate Filter (DPF) and Selective Reduction Catalyst (SCR) in order to fulfill the EU6 emission legislation.

In this paper the Engine Dynamics Library (EDL) is presented. The library is implemented in Modelica and consists of mean-value models of standard engine components. The focus of the model has been on capturing the transient engine response and the engine outlet conditions as these features are important for the total engine and EATS optimization. Comparison results between test cell measurements and simulation results of a 13 liter Volvo truck engine certified for the Post New Long-Term (PNLT) emission legislation, introduced in 2009, are presented. The Engine Dynamics Library is a new commercial library offered by Modelon.

2 Engine modeling in Dymola

Today several tools exist in which physical or semi-physical models can be implemented. Dymola [3], which is based on the open standard Modelica language, was chosen as the tool for developing an engine model library. The main reasons for choosing Modelica are the flexibility, expressiveness and openness of the language, as compared to domain specific tools, and the possibility to extend tools and libraries with in-house IP and know-how. Others have demonstrated that Modelica is suitable for engine modeling [4, 5], but the focus has not been on gas exchange modeling or predicting the exhaust gas temperature entering the EATS.

The following sections describe EDL and the component and medium models that have been implemented.

2.1 Library structure

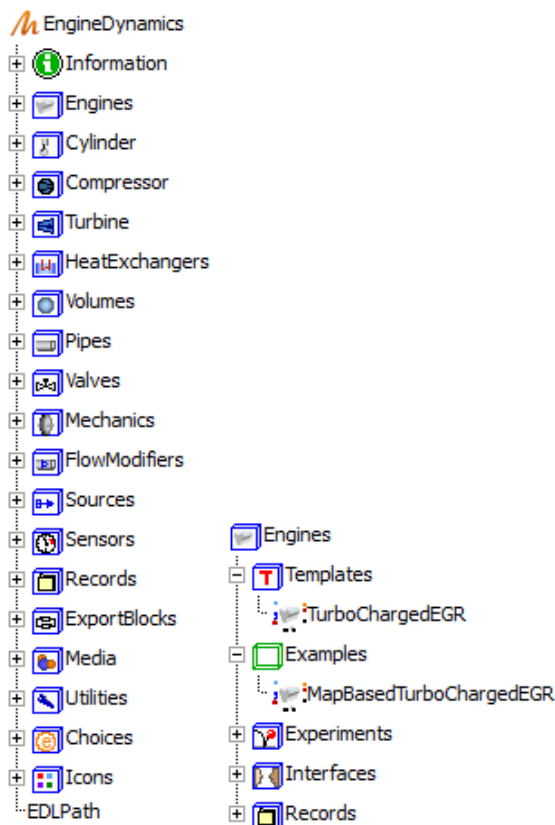


Figure 1: EDL and sub packages (left), Engines package (right)

The structure of EDL is shown to the left in Fig. 1. The library is divided into packages for each physical component, plus some additional packages for supporting components and classes. There is also a package named *Engines*, shown to the right of Fig. 1,

which contains examples of configured engine models and experiments.

EDL is not based on the Modelica.Media or Modelica.Fluid packages. Medium property models and base classes for fluid systems modeling are based on classes in the Modelon Base Library, which is delivered with EDL. EDL share base classes with Modelon's Liquid Cooling Library (LCL), Heat Exchanger Library (HXL) and Vehicle Dynamics Library (VDL), making them all compatible. The libraries can be used together for different kinds of vehicle analysis, for example EDL, LCL and HXL together forms a powerful solution for thermal management analysis, and EDL and VDL can be used together for drivability analysis.

2.2 Cylinder

The cylinder component (Fig. 2) is based on a mean value combustion model as described in [6]. The component boundary conditions are boost pressure and temperature, exhaust manifold pressure, engine speed, fuel injection and other control signals. The empirical correlations described in the following sections (often 2-dimensional maps) can easily be replaced by any equation based models, for example simple qualitative models found in literature, regression models or neural network models.

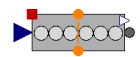


Figure 2: Cylinder

Flow model

The cylinder mass flow is modelled by means of a volumetric efficiency defined as:

$$\dot{m}_{charge} = \rho_{in} \cdot \lambda(p_{Boost}, \omega_e) \cdot \frac{V_d}{N} \cdot \frac{\omega_e}{2\pi} \quad (1)$$

where λ is the volumetric efficiency, V_d is the displaced volume, N is the number of revolutions per cycle, p_{Boost} is the inlet manifold pressure and ω_e is the engine rotational speed. $\lambda(p_{Boost}, \omega_e)$ is modelled by a two-dimensional map obtained from measurements.

Torque model

For the torque model we define brake mean effective pressure, p_{me} and fuel mean effective pressure, $p_{m\phi}$, as:

$$p_{me} = \frac{T_e \cdot 4\pi}{V_d} \quad p_{m\phi} = \frac{H_l \cdot m_\phi}{V_d} \quad (2)$$

where T_e is the engine torque, H_l is the fuel lower heating value and m_ϕ is the mass of fuel burnt per

combustion cycle. The engine efficiency can then be written as:

$$\eta_e = p_{me} / p_{m\phi} \quad (3)$$

Following the Willans Approximation [6], a torque model on the following form is implemented:

$$p_{me} = e(p_{m\phi}, \omega_e) \cdot p_{m\phi} - p_{me0f}(\omega_e) - p_{me0g} \quad (4)$$

where the energy conversion efficiency, e is modelled by a two-dimensional map obtained from measurements, the mechanical friction, p_{me0f} , is mapped from engine speed, p_{me0g} , is the cycle-averaged pressure difference between inlet and exhaust manifolds.

Exhaust gas properties

The outlet exhaust gas temperature is mapped from engine speed and injected fuel. The transferred heat to the cylinder block is obtained from energy balance over the component boundaries.

The composition of species in the exhaust gas is modelled by a stoichiometry matrix for the combustion. Complete combustion of the injected fuel is assumed. The NO_x and soot generation is modelled by a regression model [7] on the form:

$$y(t) = \phi^T \theta + e(t) \quad (5)$$

where $y = (C_{NO_x}, C_{Soot})^T$ are the NO_x and soot concentrations of the exhaust gas, the regressor $\phi = (1, u_1, u_1^2, \dots, u_1^N, u_2, \dots)^T$ contains the first and higher order terms of the following signals:

- Injected fuel amount, m_f
- Fuel injection timing, ζ
- Needle opening angle (controls the fuel injection pressure), β_f
- CO_2 concentration in the inlet manifold, C_{CO_2}
- Inverse stoichiometric air to fuel ratio, λ^{-1}
- Engine speed, ω_e

θ are the model parameters and e is the model error. In the experiment described in section 3.2, all of the input signals to the model come directly from model control signals or boundary conditions, except for the inlet manifold CO_2 concentration and air to fuel ratio. These variables are simulated in the engine system model and the simulated values are used as inputs to the emission model.

2.3 Compressor and turbine

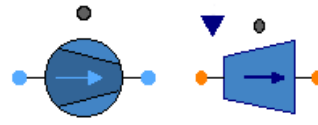


Figure 3: Compressor and VGT

The compressor and variable geometry turbine (VGT) components (Fig. 3) are both parameterized by maps for mass flow rate and isentropic efficiency. The components model a polytropic thermodynamic process with mechanical power crossing the component boundary via a rotational mechanical flange. Quasi-static balance equations for conservation of substance mass and energy are used, i.e. storage of mass and energy is not considered and the outlet properties respond instantly to property changes of the inlet flow. These equations assume:

- The amount of mass inside the component is small compared to that in the upstream and downstream pipes, which is covered by volume components connected to these components.
- The heat capacity of the solid parts are lumped together with the wall heat capacities of the volume components connected upstream and downstream of these components.
- The rotational kinetic energy of the solid parts is modeled by a separate inertia component connected to the rotational flange connector of these components.

The mapped isentropic efficiency, η_{is} , defines the deviation from an isentropic process [8].

$$\eta_{is} = \frac{h_{out, isentropic} - h_{in}}{h_{out} - h_{in}} \quad (Compressor) \quad (6)$$

$$\eta_{is} = \frac{h_{out} - h_{in}}{h_{out, isentropic} - h_{in}} \quad (Turbine) \quad (7)$$

where h_{in} is the inlet specific enthalpy, h_{out} is the outlet specific enthalpy and $h_{out, isentropic}$ is the outlet specific enthalpy of an isentropic process.

The variable geometry turbine is modeled using several maps of isentropic efficiency and mass flow rate for different positions, the properties are interpolated linearly between the mapped geometry settings. The turbine model currently contains no compensation for the upstream pressure oscillations. Internal losses

from heat transfer to the housing and mechanical friction are currently modeled as a constant efficiency factor. The turbo moment of inertia is captured by a separate inertia component connected between the compressor and turbine components in the engine system model.

2.4 Heat exchangers

A quasi-static heat exchanger model with table based efficiency is implemented in EDL. It does not contain storage of mass or energy and the outlet fluid properties respond instantly to inlet property changes. The component has interchangeable friction models with different levels of detail for the primary and secondary flow channels. A model on the following form was chosen because it is easily calibrated to fit measured data:

$$dp = f \cdot \frac{\rho}{2} \cdot v^n \quad (8)$$

Here dp is the pressure drop over the channel, f is the friction factor, ρ is the fluid density, v is the flow velocity and n is a constant. Note that for $n = 2$, this corresponds to the Darcy-Weisbach equation for pressure loss due to friction in a pipe. The constants f and n are chosen to fit measurement data.

The heat transfer is modeled by defining heat exchanger efficiency as $\varepsilon = Q/Q_{max}$. The maximum transferable heat Q_{max} is calculated from the heat capacity flow and inlet temperatures of the two channels. The model is parameterized by specifying a two-dimensional map for the efficiency from the mass flow rates in the two channels.

2.5 Volumes

All fluid mass and energy storage is modelled in volume components by dynamic mass and energy balance equations. An ideal mixture is assumed and a number of different components are available, which have different port configurations. The volume models have the option to consider wall heat capacity, heat transfer between fluid and wall (constant heat transfer coefficient model) and heat transfer to the surroundings. There is a special volume model for the inlet manifold that can handle incoming flow in a different medium model representation by mapping the fluid species between the mass fraction vectors of the two medium models. This is necessary if separate models

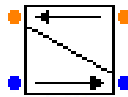


Figure 4: Heat exchanger

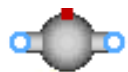


Figure 5: Two port air volume

for air and exhaust gas are used. Outgoing flows from the volume carry the average medium properties of the total volume.

2.6 Pipes

The pipe models provided in the library consider pressure drop due to friction and optionally also heat transfer effects. Several friction models can be chosen, but also here eq. 8 is used. The heat transfer model is interchangeable as well, with the options: 1) Constant heat transfer coefficient, 2) Dittus-Boelter correlation for forced convection in turbulent flow (Coefficients can be adjusted by the user). Optionally a dynamic momentum balance can be used.

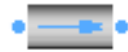


Figure 6: Air pipe model

2.7 Valves

There are a number of valve models available in EDL. The first one is designed to be easily parameterized from measured data. It defines a flow equation for the fully opened setting as eq. 8. The valve characteristics are represented by means of a relative open area that is governed by the actuation signal. Linear, quadratic and tabulated characteristics are available. The second one is implemented according to the IEC 534/ISA S.75 standards for valve sizing. It accounts for fluid compressibility effects, as well as choked conditions. For the engine model presented in this paper, the first model is used because it is easier to parameterize from measurements and choked conditions do not occur under normal operation.



Figure 7: Valve model

A butterfly type valve model has been implemented as well, including flap mechanism, torque generation on the flap by the gas flow and mechanical friction.

2.8 Medium models

The medium property models are implemented as replaceable packages with high flexibility, similar to that of the Modelica.Media package. Ideal gas mixtures based on the NASA coefficients [9] can be created and used.

In addition to this, a simplified medium model assuming a linear function for specific heat capacity of temperature, $C_p(T)$, has been implemented for performance reasons. By definition, the specific enthalpy function, $h(T)$, will become quadratic in temperature

under this assumption. In static component models, the upstream temperature $T(h)$ is calculated from the specific enthalpy of the inlet fluid connector. An explicit function for this calculation greatly improves simulation performance for system models with several such components, as the non-linear systems of equations can be reduced or completely avoided. The medium models are compatible, so all component models can carry any of the medium model types. Available in EDL are some pre-defined mixtures, used as air or exhaust gas models. The components included are CO_2 , H_2O , O_2 , N_2 and Ar for both NASA and linear $C_p(T)$ models. Also a single component dry air model is provided. To model emissions, some pre-defined exhaust gas mixtures include trace components for NO_x , $Soot$, HC and CO . The trace components are assumed to be carried by fluid flow but don't affect the thermodynamic properties of the fluid.

2.9 Mechanical

Basic rotational mechanical components are available in EDL, such as inertia and ideal gear models. The mechanical connectors of the turbo components and cylinder component are compatible with the mechanical components in the Modelica Standard Library.

3 Engine system model

A 13 liter Volvo engine certified for the Post New Long-Term (PNLT) emission legislation has been modeled using EDL. The engine is equipped with variable geometry turbine, exhaust gas recirculation governed by a valve, throttle, EGR cooler, intercooler and unit injectors. The purpose of the simulation model is to perform similar experiments that are performed in engine test cells, where the engine is mounted to an electrical dynamometer which directly controls the engine speed.

3.1 Model description

The engine system model is configured as shown in Fig. 8. The upper left connector is the air inlet connector that should be connected externally to a component defining air temperature and pressure boundary conditions. The components in the air path are connected to represent the engine system design, indicated with light blue in the figure. First there is a pipe component modeling the pressure drop over the air filter (1). Then follows compressor (2), intercooler (3) and throttle (4)

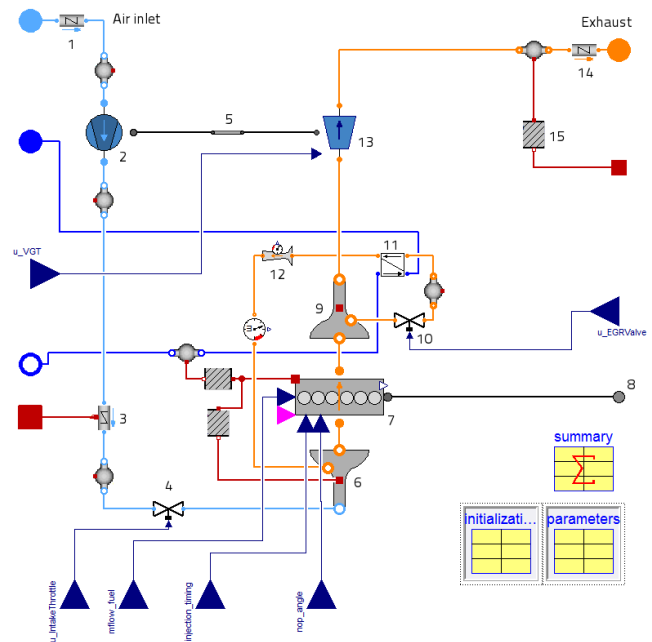


Figure 8: Engine system model with: Air filter (1), Compressor (2), Intercooler (3), Throttle (4), Turbo inertia (5), Inlet manifold (6), Cylinder block (7), Drive shaft (8), Exhaust manifold (9), EGR valve (10), EGR cooler (11), Venturi (12), VGT (13), Muffler (14), Heat transfer (15, and more)

components, each separated by volume components. The compressor is connected to an inertia model (5) that is also connected to the VGT component (13). The throttle in the lower left is connected to the inlet manifold component (6). This is a volume model that also accounts for the thermal mass of the wall and heat transfer between the gas and wall. The inlet manifold has two more connectors for gas (orange). One is connected to the cylinder block and the other is the inlet for EGR gas.

The cylinder block (7) has a rotational connector for the drive shaft that is connected to an external connector to the right in the figure (8). It is also possible to enable a support connector for the reactive torque, but it is not used here. There are real input signal connectors for injected fuel, injection timing and needle opening angle. The exhaust gas port is connected to the exhaust manifold (9), which is also a volume model including thermal mass of the wall. There is an outlet port for the exhaust gas recirculation path that is connected directly to the EGR valve (10). This is connected to a volume and then to the EGR cooler (11) and venturi (12). The venturi component is a pure sensor model that does not affect the gas flow rate or properties. The EGR gas path is then fed back to the inlet manifold.

For each volume model there is a unique pressure and temperature state introduced. As a consequence of the model layout the flow through the EGR valve is calculated from the pressure difference between the exhaust manifold and EGR volume components. The pressures are calculated during model simulation by means of numerical integration.

The exhaust manifold is also connected to the turbine component (13). Additionally, the turbine has an input signal for varying the geometry, a rotational flange connector and an outlet gas connector. The turbine component calculates a torque that is generated on the flange. Thus, the turbo rotational speed is obtained during simulation by integration of the dynamic momentum equation introduced in the inertia component, with torque terms from the turbine and compressor components. After the turbine the gas is fed to a volume model and then a pipe model that accounts for the pressure drop over the muffler (14).

The volume model in the exhaust path has a thermal connector (red square) that holds the wall temperature of the exhaust pipe. This is connected to a heat transfer component (15) that contains a linear heat transfer equation. This is also connected to an external heat connector where the ambient temperature should be provided as boundary condition. Such heat transfer components are also used to cover heat transfer between the cylinder block and coolant water, and between cylinder block and inlet manifold. The coolant path is indicated with dark blue connections. The set of connector variables in the air, gas and water connectors are identical. Only the color differ for a clearer visual model representation.

3.2 Simulation model

The engine model described above can be used in various simulation models or virtual experiments. Simulation models are created by instantiating the engine model and assigning values or signal to boundary conditions and input control signals. The following signals from the engine electrical control unit (EECU) are set as input signals:

- Injected fuel, injector timing and needle opening angle (controls the fuel pressure)
- VGT, EGR valve and throttle positions

The following physical boundary conditions are set:

- Engine coolant temperature and mass flow rate
- Ambient air temperature and pressure

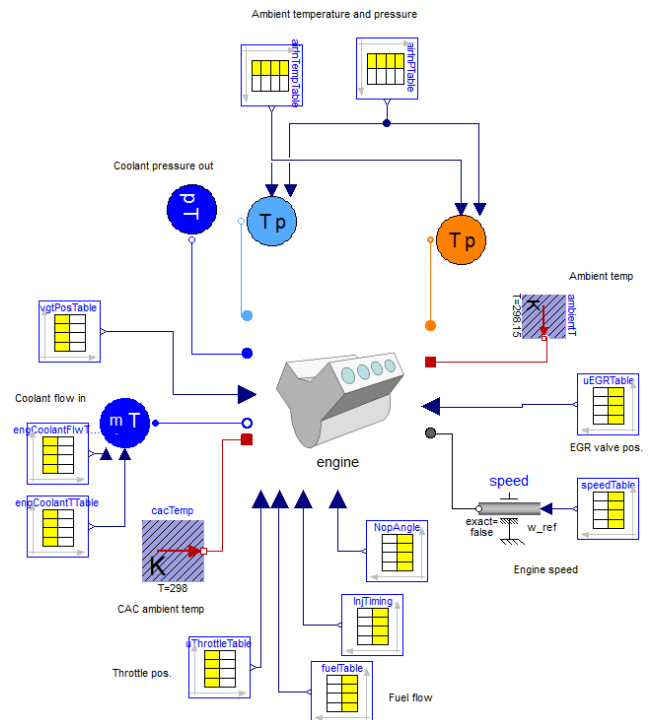


Figure 9: Simulation model of the engine in a test cell. The engine component corresponds to the engine model as shown in Figure 8.

- Engine driveshaft speed

This experiment is set up in Dymola, as shown in Fig. 9. The centered engine icon represents the engine model as shown in Fig. 8. The components with table icons are used to read signals from the engine test cell measurements from an external file. The engine component need not be connected directly to source components, but could be used in larger system models together with drive line, vehicle dynamics, coolant system or exhaust after treatment system models.

4 Calibration

The calibration is done component by component. The benefit with this approach is that it is possible to change a component and only recalibrate the new component without needing to recalibrate the whole systems. Validation is performed both component by component and for the overall system using steady-state and dynamic data. The exhaust gas thermal dynamics is calibrated using an exhaust gas path subsystem model.

4.1 Static correlations

The calibration of the static engine correlations is performed in Matlab using steady state measurement data. Flow model parameters for pipe, valve and heat exchanger models are calibrated with a Least-Squares method using static engine screening data. For the compressor and VGT the maps supplied by the manufacturer were used. Heat exchanger measurements were also supplied separately, and not identified from the screening data. The maps for energy conversion efficiency, volumetric efficiency and exhaust gas temperature used in the cylinder component were calibrated using the surface fitting tool gridfit [10]. The calibration data for this component consisted of a partial load map collected from an engine test cell. For the valves, one dimensional look-up tables for relative open area from the control signal were created. Some results from the calibration procedure are presented in the following figures. Fig. 10 shows the fitted surface for volumetric efficiency together with measurements. Fig. 11 shows the measured mass flow rate through the intercooler at different pressure drops together with a calibrated model using equation 8. Fig. 12 shows the fitted look-up table for throttle relative open area.

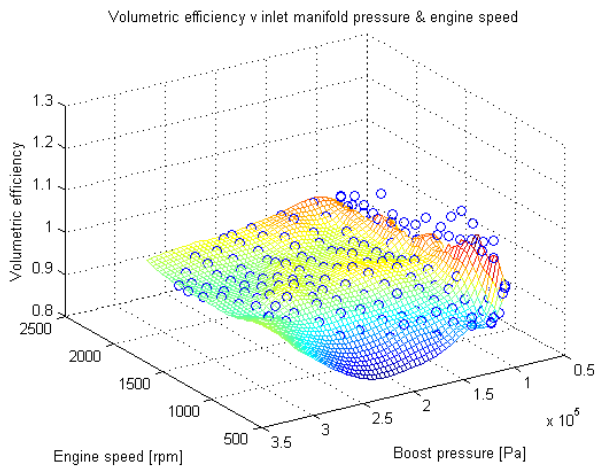


Figure 10: Volumetric efficiency map, fitted map and measured data

4.2 Emission model

The linear regression model is calibrated by least squares estimation [7]. For calibration, the initial 10 minutes of the dynamic JE05 cycle, further described in section 5, were used. The remaining 20 minutes are then used for validation of the calibration result. The

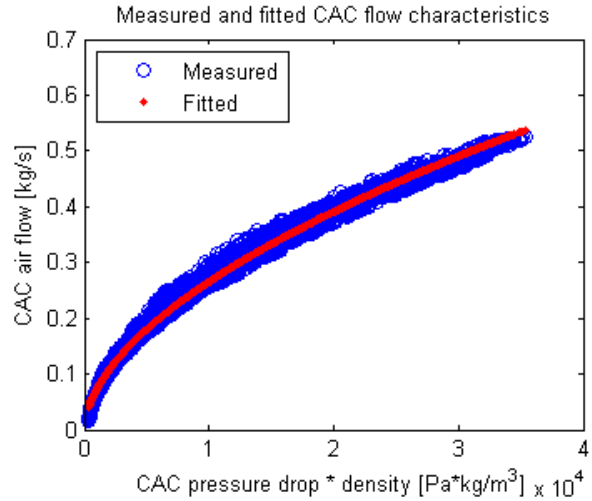


Figure 11: Intercooler flow friction model

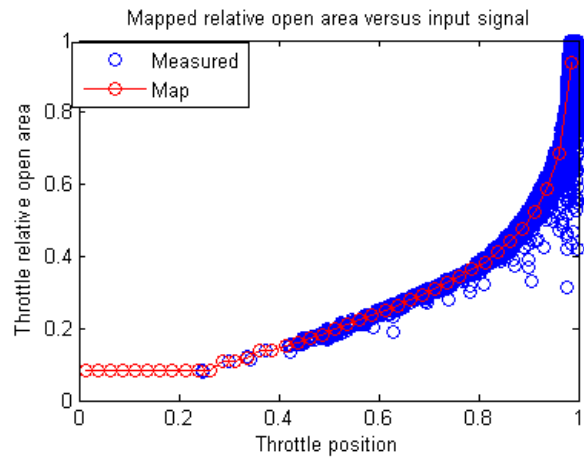


Figure 12: Throttle relative open area

following regressor found to best best result

$$\phi = (1, m_f, m_f^2, m_f^3, \zeta, \zeta^2, \zeta^3, \beta_f, \beta_f^2, \beta_f^3, C_{CO2}, C_{CO2}^2, C_{CO2}^3, \lambda^{-1}, \lambda^{-2}, \lambda^{-3}, \omega, \omega^2, \omega^3) \quad (9)$$

4.3 Parameter optimization in JModelica.org

JModelica.org [11] has been used for optimization of model parameters for heat transfer and thermal dynamics in the exhaust gas path. The method used is the derivative free Nelder-Mead simplex method [12, 13]. Derivative free methods do not require that the model provides derivatives of the objective function with respect to tuner variables. That makes them well suited

for optimization of more complex models, and model modifications for optimization purposes are not necessary. The following parameters were optimized to obtain the best possible result for the exhaust gas temperature during transient cycles:

- Thermal conductance between exhaust gas and wall
- Heat capacity of the exhaust pipe wall
- Thermal conductance between the wall and the surrounding air

The dynamic exhaust gas temperature response, presented in Fig. 23, is very different from the instantaneous outlet gas temperature from the quasi-static VGT model. This is both due to thermal mass of the metal parts, and heat transfer to the surrounding air. The heat capacity and thermal conductances mentioned above model the dynamic exhaust temperature response from the VGT outlet temperature. The initial 10 minutes of the JE05 cycle were used for parameter optimization. The remaining 20 minutes are then used for validation of the calibration result.

5 Validation

The models have been validated, both by individual component experiments, and by complete engine system simulation. The used data was collected from an engine test cell and consisted of partial load map data and of the Japanese emission cycle, JE05. The JE05 cycle is one of the legislation requirements in the Post New Long-Term (PNLT) legislation.

5.1 Turbo model validation

The turbo model with rotational speed as dynamic state was validated separately with boundary conditions from a partial load map. An experiment model is set up where a compressor and VGT component are connected with an inertia model in between. Upstream and downstream pressure and temperatures and VGT position are prescribed and the resulting mass flow rate, outlet temperature and rotational speed are validated for the compressor and turbine models. Fig. 13 shows a comparison of the turbo flow rates. In Fig. 14 the turbo model outlet temperatures are shown.

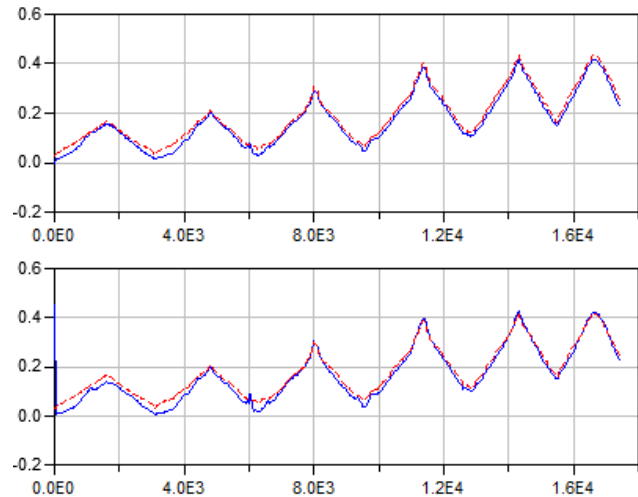


Figure 13: Turbo model validation. Top: Exhaust flow rate [kg/s], simulated (solid) and measured (dashed). Bottom: Air flow rate [kg/s], simulated (solid) and measured (dashed)

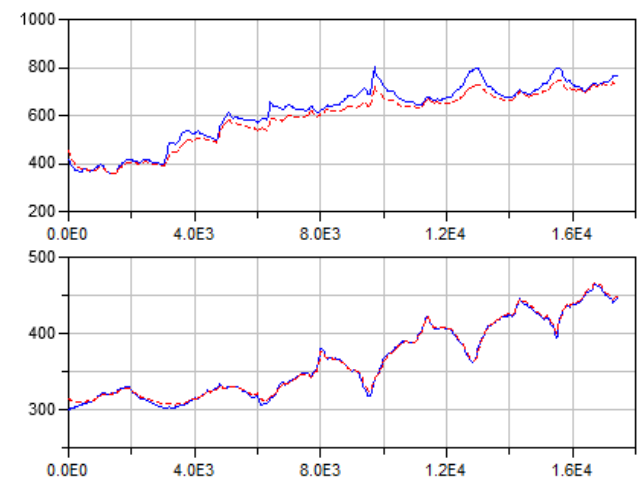


Figure 14: Turbo model validation. Top: Turbine outlet temperature [K], simulated (solid) and measured (dashed). Bottom: Compressor outlet temperature [K], simulated (solid) and measured (dashed)

5.2 EGR model validation

The EGR valve model is validated with part load map data. Upstream and downstream pressures are prescribed and the simulated EGR flow rate is compared to measurements. The result is presented in Fig. 15.

5.3 Verification of non-minimum phase and sign reversal

An engine equipped with VGT and EGR valve has some essential system properties such as non-minimum phase behavior in the intake manifold pressure and a non-minimum phase behavior and a sign re-

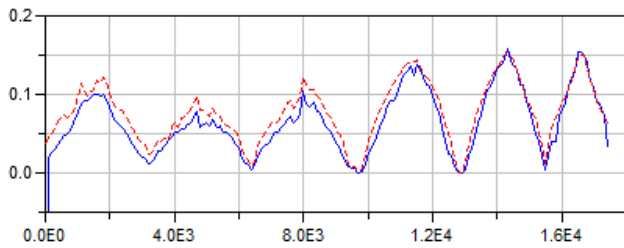


Figure 15: EGR flow model validation. EGR flow [kg/s], simulated (solid) and measured (dashed)

versal in the compressor flow [15]. Fig. 16 shows that the the model captures the non-minimum phase behavior between the EGR valve position, u_{egr} , change and inlet manifold pressure, p_{in} .

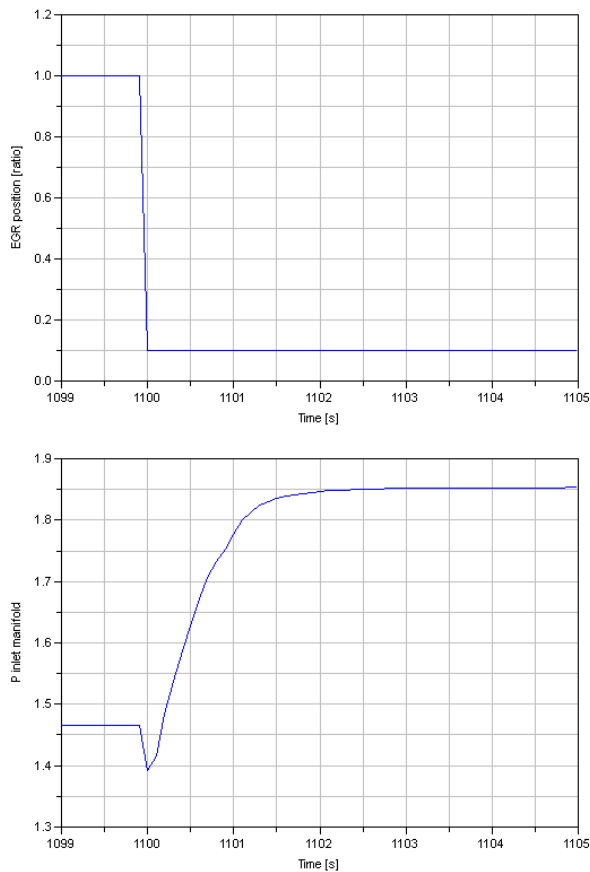


Figure 16: Dynamic verification of the non-minimum phase between u_{egr} and p_{in} using steps. Operating point: $\omega_e = 1500$ rpm, $T_e = 670$ Nm, $u_{vgt} = 0.5$ ratio.

Fig. 17 shows that the model capture the non-minimum phase behavior between the VGT position, u_{vgt} , and the compressor mass flow \dot{m}_c . Notice that initially the DC gain between u_{vgt} and \dot{m}_c is negative but after a while it becomes positive. This phenomena is even better seen in Fig. 18 where the u_{vgt} is

slowly changed from complete opened vanes towards closed position. As the sweep is performed slowly and the other operating conditions are kept constant, the results can be regarded as steady state results.

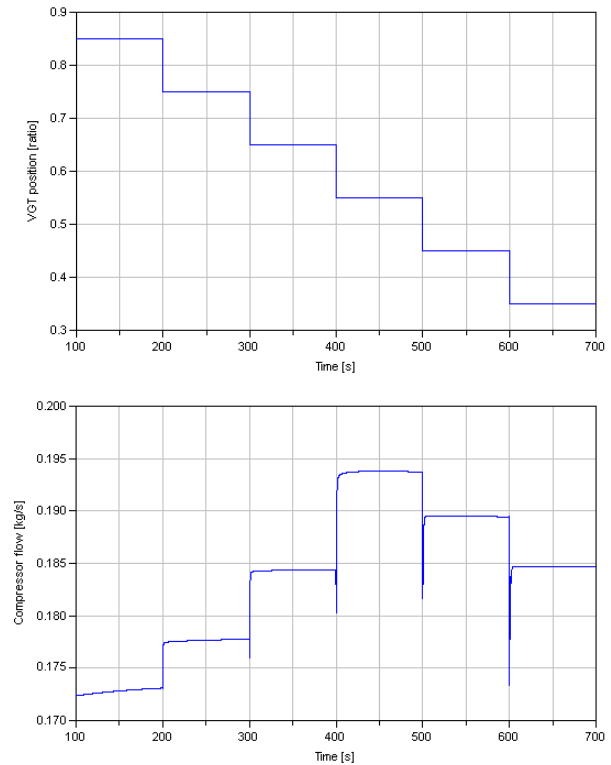


Figure 17: Dynamic verification of the non-minimum phase between u_{vgt} and \dot{m}_c using steps. Operating point: $\omega_e = 1500$ rpm, $T_e = 670$ Nm, $u_{egr} = 1$ ratio.

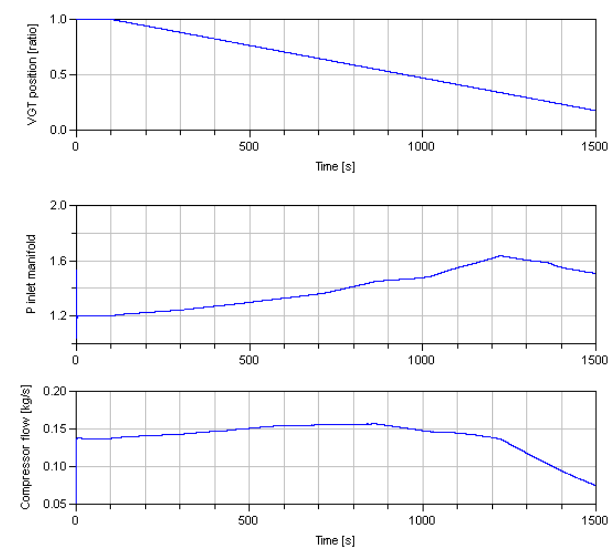


Figure 18: Slow sweep of the u_{vgt} from fully open to closed position. Operating point: $\omega_e = 1500$ rpm, $T_e = 670$ Nm, $u_{egr} = 1$ ratio.

5.4 Dynamic validation

The engine system model is validated with JE05 boundary conditions using the experiment setup in Fig. 9. The cycle is 1830 seconds long and the simulation time for the whole cycle was 735 seconds (2.5x faster than real-time) on a standard laptop. The JE05 is a very transient cycle which contains mostly city driving with some high way driving. The engine speed variations during the complete cycle are shown in Fig. 19 and the load variations are shown in Fig. 20.

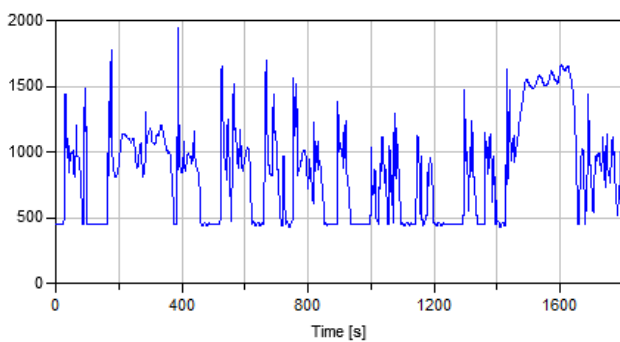


Figure 19: JE05 engine speed [rpm]

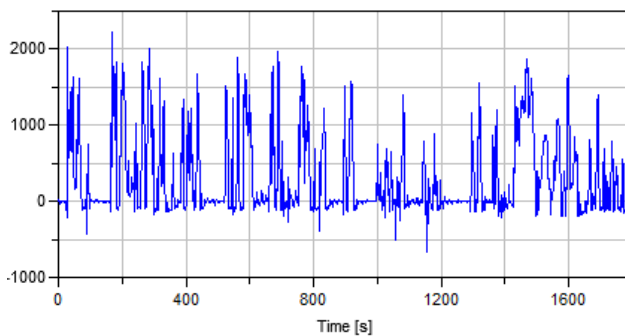


Figure 20: JE05 engine torque [Nm]

The resulting full cycle exhaust gas temperature is shown in Fig. 21 and NO_x emissions are shown in Fig. 22. Both the modeled exhaust temperature and the NO_x emission captures most of the behavior. The modeled exhaust temperature differs from the measured temperature in the end of the JE05 cycle. The temperature before the VGT capture the temperature behavior correct also in the end of the cycle this indicate that there are still heat transfer effects that need to be incorporated in the model.

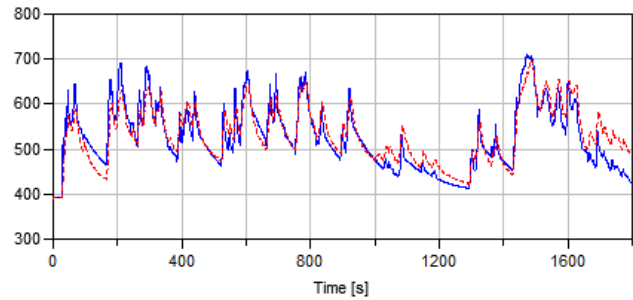


Figure 21: Complete model validation. Exhaust gas temperature [K], simulated (solid) and measured (dashed)

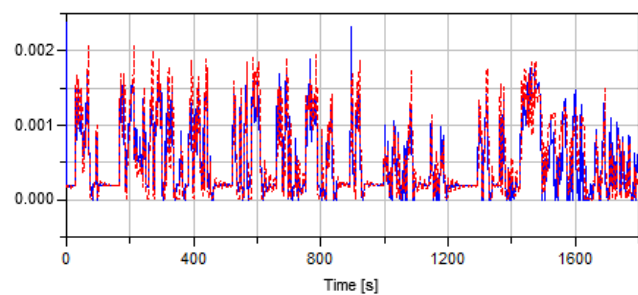


Figure 22: Complete model validation. Exhaust NO_x concentration [kg/kg], simulated (solid) and measured (dashed)

Figures 23 - 27 show simulation results for engine torque, mass flow rates and exhaust gas temperature from a part of the cycle (750 - 1000 s). The exhaust gas temperature is measured in the pipe 1 meter after the turbine. As can be seen in Fig. 23 the model captures most of the behavior. Figures 24 - 26 show that the model captures the dynamics of the exhaust, EGR and air mass flows. The ERG flow in Fig. 25 shows a small time lag of the measured flow compared to the simulated. This is likely due to a time lag in the EGR flow sensor.

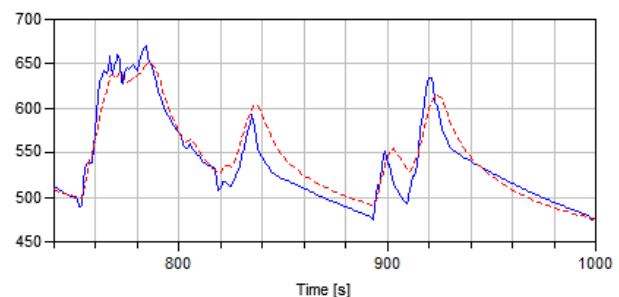


Figure 23: Complete model validation. Exhaust gas temperature [K], simulated (solid) and measured (dashed)

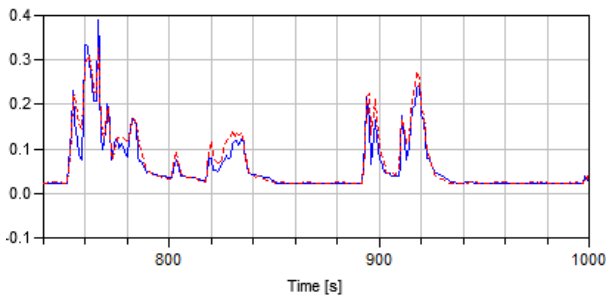


Figure 24: Complete model validation. Exhaust gas flow rate [kg/s], simulated (solid) and measured (dashed)

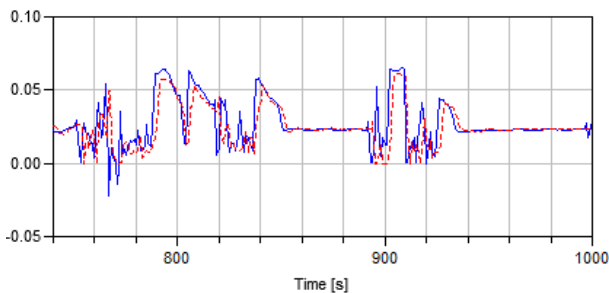


Figure 25: Complete model validation. EGR flow rate [kg/s], simulated (solid) and measured (dashed)

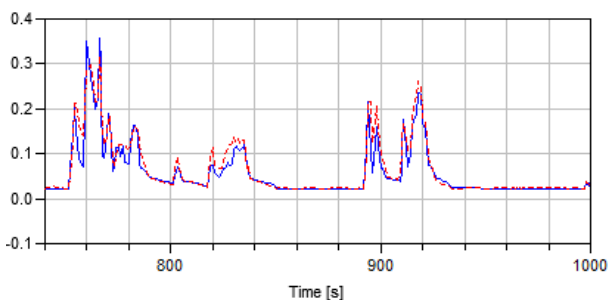


Figure 26: Complete model validation. Air flow rate [kg/s], simulated (solid) and measured (dashed)

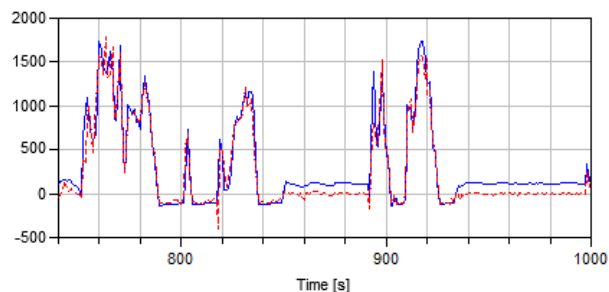


Figure 27: Complete model validation. Engine torque [Nm], simulated (solid) and measured (dashed)

The model captures most of the dynamics of the engine torque, but for the idling part (e.g. 850-890s) there is an offset between modeled and measured

torque (Fig. 27). The difference may be explained by the fact that the friction or the pumping loss measurements which the model is based on are not correct in this region.

Fig. 28 shows the NO_x emissions. The black-box model succeeds to capture the behavior. The NO_x levels are quite close to the measured level in steady state operation, and the peaks are often quite close to the measured level regarding timing and level. The NO_x level was measured by a Horiba system, which isn't capable of measuring fast transients and the measurements can be regarded as a filtered values.

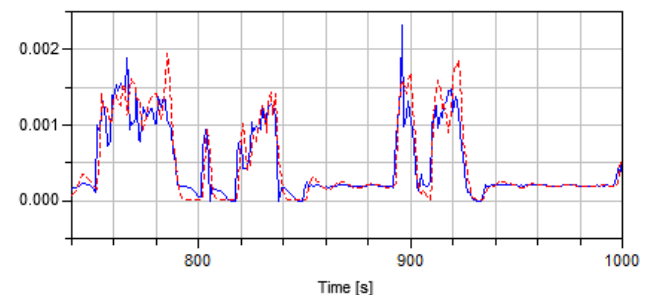


Figure 28: Complete model validation. Exhaust NO_x concentration [kg/kg], simulated (solid) and measured (dashed)

6 Discussion

The components in the 13L Volvo PNLT engine are primarily modeled by a physical first-principle approach. The selected inputs for the emission model does not capture the effect of the wall temperature and a next step can be to parametrize a cylinder wall temperature model in order to model the effects of cold starts. The current simple emission model captures most of the transient effects and in order to further improve the transient optimization based on the models the accuracy needs to be improved. Instead of assuming CO₂ in the exhaust manifold based on stoichiometric combustion, it can be added as an output of the emission model. This may improve the estimation of the CO₂ in the inlet manifold which is one of the inputs to the emission model. There exists several data driven emission models with similar computational complexity that would be interesting to compare against [16]. The plan for the future is that EDL will be expanded with more combustions model options, including cycle-resolved in-cylinder behavior. By introducing the effects of pressure pulses and improving the internal loss model, the turbo model can also be further improved.

JModelica.org was used for the optimization of parameters for the heat transfer and thermal dynamics and Dymola was used to export the FMU model. JModelica has extended the Modelica language for increased optimization functionality. The derivative-free simplex method used worked very well for parameter optimization for a model of this complexity without requiring any model modifications. Other tools also exist that can perform calibration using similar methods, for example the model calibration feature in Dymola or Isight. Isight was also tested for the same optimization task and the simplex method available there gave equivalent results to JModelica regarding optimization time and result.

The simulation speed is about 2.5 times faster than real-time using the Dymola integrated Radau solver. This is a variable step-length solver, and the fast average simulation speed does not guarantee that the current model can be used in applications with hard real-time requirements, but this was not in the scope for this model. For hard real-time simulations, fixed-step solvers must be used. This introduces harder requirements on the model regarding fast dynamics and function evaluation time.

As the models of the PNLT engine managed to capture the engine out conditions and the dynamical behavior in the air gas path, the model can be used to develop engine control strategies that reduce the requirement on the EATS. With transient control strategies that reduce transient emissions, the EATS volumes (e.g. DOC, DPF and SCR) may be reduced. Also by adapting the engine control strategies based on the condition of the EATS (e.g. temperature, aging and poisoning) the EATS volumes may be reduced. The fuel cost of the different engine control actions depends significantly on the engine hardware and each has an optimal trade-off between fuel cost and product cost. Engine models based on EDL together with a SIL environment which includes the control strategies is a powerful approach in the investigation of finding the optimal trade-off.

7 Conclusions

In this article it has been demonstrated that the newly developed Engine Dynamics library and Dymola can be used for simulation of the gas exchange, transient flow and temperatures and emission trends for a 13L Volvo PNLT engine. All components and parameters have been calibrated component wise without any global compensation. Calibration data comes from

an engine screening where measurements are made to isolate the different components. Therefore a component can be replaced without any need of a new complete engine screening, allowing for virtual prototyping of new concepts. This is an important advantage compared to black-box modeling of the complete engine, which would require a complete new screening when changing a single component. Finding parameter values for the heat transfer and thermal dynamic in the exhaust that matches measurements is an optimization problem that has been solved using JModelica. The parameters were successfully optimized resulting in good estimation of the exhaust temperature dynamics. The models captured the essential system properties in the gas exchange such as non-minimum phase behavior and sign reversal. As the exhaust mass flow, exhaust temperature and emissions were shown to be well captured the model can be used in order to evaluate control strategies of the air gas management and to find a trade-off between fuel-economy, transient response, engine emissions and EATS requirements. The system identification of the NOx emissions gave good results in the operating area of the JE05 cycle and captured the trends. This indicates that the selected inputs to the emission model contain most of the entities that affect the emissions. Using variable step-length solvers, the engine model simulates faster than real-time for the JE05 cycle. This is a very transient cycle, and therefore the expectation is that other transient cycles will also simulate with real-time like simulation times.

References

- [1] T. Johnson *Diesel Emissions in Review*, SAE Technical Paper 2011-01-0304, 2011.
- [2] R. Cloudt and F. Willems. *Integrated Emission Management strategy for cost-optimal engine-aftertreatment operation*, SAE Technical Paper 2011-01-1310, 2011.
- [3] Dymola User Manual, Volume 1, Lund, 2011
- [4] J. Batteh, M. Tiller and C. Newman. *Simulation of Engine Systems in Modelica*, Proceedings of the 3rd Modelica Conference, Linköping, Sweden, 2003.
- [5] A. Picarelli and M. Dempsey. *Investigating the Multibody Dynamics of the Complete Powertrain System*, Proceedings of the 7th Modelica Conference, Como, Italy, 2009.

- [6] L. Guzzella and C.H. Onder. *Introduction to Modeling and Control of Internal Combustion Engine Systems*, 2nd edition, 2010. ISBN 978-3-642-10774-0.
- [7] R. Johansson. *System modeling & Identification*, 2009. ISBN 0-13-482308-7.
- [8] HIH Saravanamuttoo, GFC Rogers and H Cohen. *Gas Turbine Theory, Fifth Edition*, 2001. ISBN 978-0-13-015847-5.
- [9] B.J. McBride, M.J. Zehe and S. Gordon. *NASA Glenn Coefficients for Calculating Thermodynamic Properties of Individual Species*. NASA report TP-2002-211556, 2002.
- [10] J. D'Errico. *Understanding GRID-FIT*, 2006. Available for download at <http://www.mathworks.com/matlabcentral/fileexchange/8998> (last accessed 20120228).
- [11] J. Åkesson, K-E. Årzén, M. Gäfvert, T. Bergdahl and H. Tummescheit. *Modeling and Optimization with Optimica and JModelica.org - Language and Tools for Solving Large-Scale Dynamic Optimization Problems*, Computers and Chemical Engineering, 34:11, pp. 1737-1749, November 2010
- [12] S. Gedda. *Calibration of Modelica models using derivative-free optimization*, Master's thesis 2011:E46, Lund University, Faculty of Engineering, Centre For Mathematical Sciences, Mathematics, 2011.
- [13] S. Gedda, C. Andersson, J. Åkesson and S. Diehl. *Derivative-free Parameter Optimization of Functional Mock-up Units*. In 9th International Modelica Conference, 2012.
- [14] MODELISAR(07006). *Functional Mock-up Interface for Model Exchange* Available for download at: <http://www.functional-mockup-interface.org/> (last accessed 20120228).
- [15] J. Wahlström and L. Eriksson. *Modeling of a diesel engine with VGT and EGR capturing Sign Reversal and Non-minimum Phase Behavior*. Proceedings of the Institution of Mechanical Engineers, Part D, J. of Automobile Engineering, Volume 225, Issue 7, July 2011.
- [16] M. Grahn and T. McKelvey. *MA Structure and Calibration Method for Data-driven Modeling of NOX and Soot Emissions from a Diesel Engine*. SAE Technical Paper 2012-XX-0351, 2012.

Library for First-Principle Models of Proton Exchange Membrane Fuel Cells in Modelica

Kevin L. Davies Christiaan J.J. Paredis Comas L. Haynes
Georgia Institute of Technology
Atlanta, Georgia USA

Abstract

This paper describes the architecture and key equations of FCSys, a library to model proton exchange membrane fuel cells (PEMFCs) in Modelica. The motivating goal of this work is to reconcile many of the published models of PEMFCs and combine them in a reconfigurable PEMFC model that is effective for a variety of uses. It is necessary to distill equations from fuel cell literature into forms that at once capture the essence of the physical interactions, are conducive to the physical modularity of the device, and work within the constraints and take full advantage of the Modelica language.

Since the behavior of PEMFCs depends on both advection and diffusion, a suitable alternative to the Modelica Fluid library and the stream concept is necessary. The proposed solution uses a “mixing” scheme based on the exponential of the Péclet numbers for each transport process. Storage and transport processes are co-located in each subregion of a rectangular grid—all in the same base model. The Onsager formulation is used, whereby the effort and flow rate are conjugates of the entropy flow rate associated with energy transfer.

The implementation is modular. It allows species to be enabled independently for each region. In addition, the geometric axes may be independently enabled (up to 3D) and shearing (transverse momentum) may be optionally included. Chemical/electrochemical interactions are communicated in a fully acausal manner through expandable connectors.

This paper focuses on the motivation, background, and approach. Future publications will describe the ongoing work to calibrate, validate, and utilize the model for particular case studies. The library is made available as open source.

Keywords: PEMFC; three dimensional; fluid dynamics; electrochemistry; heat transfer; advection; diffusion; momentum; Onsager

1 Introduction

In certain power applications, fuel cell (FC) systems are preferable because they can convert fuel energy to work more efficiently than internal combustion engines and have energy-to-power ratios that can be easily adapted, unlike batteries. A FC system can be refueled quickly like an internal combustion engine (ICE) system, or it can be designed to recharge like a battery by operating in electrolysis mode [4]. Of the various fuel cell technologies, PEMFCs are best suited to meet the power-cycling and packaging requirements of vehicles and portable devices.

However, the cost and durability, and to a lesser extent, size and weight, of PEMFCs are not yet adequate to justify their use beyond niche devices and select demonstrations. Much work is being done to investigate the modes of failure and degradation, develop new materials and structures, improve manufacturing processes, and design better systems [26]. Mathematical models of PEMFCs are being used to help understand the relevant physical phenomena, study the effects of design choices, and perform model-based control. The breadth of these goals has led to a multitude of specialized models.

PEMFCs have a solid polymer-based electrolyte (the PEM) and operate at low temperatures (typically below 100 °C). As shown in Figure 1, a single-cell PEMFC has few core components: PEM, electrodes, gas diffusion layers (GDLs), and flow plates [14]. However, most applications require a higher electrochemical potential difference than a single-cell PEMFC can provide; therefore, two or more cells are joined back-to-back to form a PEMFC stack.

A PEMFC operates on the electrochemical energy released by the reaction of hydrogen and oxygen to produce water (Eq. 1c). Its PEM (electrolyte) controls the reaction by selectively passing protons while acting as a barrier layer to hydrogen, oxygen, and electrons (see Fig. 1). This forces the reaction to occur

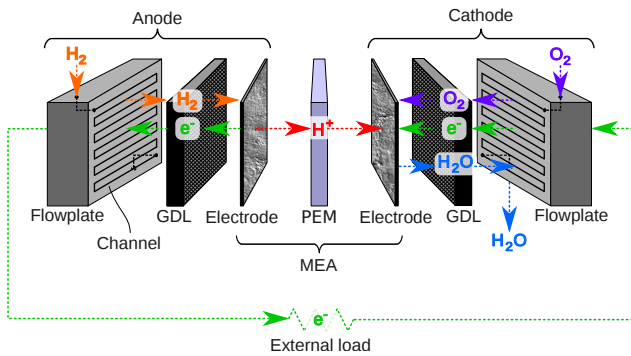
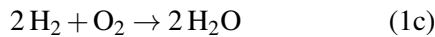
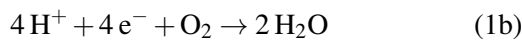
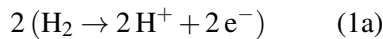


Figure 1: Layers of a single-cell PEMFC and the primary paths of hydrogen (H_2), oxygen (O_2), protons (H^+), electrons (e^-), and water (H_2O) during normal operation

in two sub-reactions: the hydrogen oxidation reaction (HOR) whereby hydrogen is consumed and protons and electrons are produced (Eq. 1a) and the oxygen reduction reaction (ORR) whereby oxygen, protons, and electrons are consumed and water is produced (Eq. 1b). In order to complete the full reaction, the electrons must traverse an external path. The path is provided by an external load which can harness the energy of the net reaction.



A broadly applicable PEMFC model library would need to contain models that are *physically representative*, meaning their predictions of behavior match reality (i.e., *accuracy*) and their structure corresponds to the physical domain. The PEMFC model library should approximate the dynamic voltage-current response of actual cells at nominal operating conditions and varying large signal electrical currents (e.g., [27, p. 3787, Figs. 2b and 2c]). It should capture the operational effects of design parameters including component sizes and material properties (for hardware analysis and design) and should be capable of linearization (for control analysis and design). It should be able to describe relevant phenomena including electrochemical reactions, chemical/electrochemical transport, heat transport, and heat generation. It should have variable *fidelity*, that is, degree of spatial, dynamic, or behavioral detail. Finally, it should be *modular*, meaning its components can be interconnected in various ways to build models of larger systems. Unfortunately, however, no current PEMFC model library can provide these features and capabilities over the required range of operating conditions.

2 Related work

For reasons elaborated later, the acausal formalism and the Modelica language in particular is ideal for a dynamic, variable-fidelity, modular, and systems-oriented model of a PEMFC. There are hundreds of published PEMFC models [28], yet most of these use computational fluid dynamics (CFD) or causal (signal-based) models. Only four acausal, dynamic, and cell-level FC models are known to have been published; three are of PEMFCs and one is of a solid oxide fuel cell (SOFC).

Rubio et al. openly shared a 1D (through-the-cell) declarative PEMFC model which includes electro-osmotic drag, double layer capacitance, variable choice of assumptions, and detailed diffusion with pore and species interactions (Knudsen flow and Maxwell-Stefan eqs.). However, the model is isothermal, does not include heat generation or a model of the flow plate, and only interacts with its surroundings electrically (no external thermal or fluid terminals) [22, 23].

Davies and Moore published a quasi-2D (through-the-cell and along-the-channel) declarative PEMFC model which includes material and heat transport and storage, electro-osmotic drag, and variable choice of assumptions. However, the models of the cell's layers are not defined in a physical manner; for example, the electrode layers do not include chemical transport (only reactions and charge transport) [7, 8]. The last published version was based on the Modelica Fluid library [9]. As a result, it raised concerns (at the 7th Modelica Conference) and had issues related to the integration of advection and diffusion, since Modelica Fluid offers a solution that is limited to purely advective flow.

McCain et al. implemented the model of McKay et al. [18] (mentioned previously) within a declarative formalism in order to linearize the model for control studies. However, the sub-models of the chemical species do not interact except in the flow plates and the PEM [17].

Salogni and Colonna published a 1D (along-the-channel) declarative model of a SOFC. It is well-constructed, but since it treats each anode-to-cathode section of the cell as a integrated unit, its modularity does not resolve the physical layers of the cell [24].

A related approach is chemical bond graphs. Bond graphs have been used for decades to chemical reactions [5] and even applied to fuel cells [3, p. 355]. They are physical (in terms of energy) and are useful to trace causality, but they are not acausal. According

to Cellier, bond graphs have not yet been successfully applied to problems in fluid dynamics. The reason is that fluid systems require mass conservation in addition to energy conservation [5, p. 331].

3 Architecture

The present model is described in differential algebraic equations (DAEs). Spatial variances are represented in terms of differences rather than derivatives. As stated by Mattiussi [15, pp. 2–3], this representation has three advantages: (i) it provides a unified perspective that is appropriate for many theories, (ii) it directly correlates the discretization of the physical region and the structural properties of the applied theories, and (iii) it is based on intuitive geometrical and physical concepts that help distinguish the numerical methods (e.g., finite difference method, finite volume method, and finite element method) from the underlying theories. In addition, powerful modeling tools (e.g., Dymola [12]) exist that can solve a model for the imposed causality, partition a dynamic model into the most numerically efficient systems of algebraic equations (i.e., resolve algebraic loops through tearing), perform index reduction (i.e., eliminate structural singularities), and linearize a model. Ultimately, this can result in a flexible and robust model that simulates quickly.

Table 1 summarizes the four base types of connectors that are used in the Modelica implementation. Figure 2 shows the hierarchy of the connectors, with the lowest level at the bottom. The flows of the material, linear momentum, and energy connectors are the rates of those quantities. The flow variable of the volume connector is the volume itself (not the volume flow rate). This allows the volume connector to be used to impose additivity of volume or Amagat’s law—that the sum of the total volume of the region is the partial volumes of the species evaluated at the total pressure [20]. The effort variable is chosen such that the product of effort and the rate of the quantity is the entropy flow rate associated with the energy transfer. This approach is convenient for representing behavior in terms of Onsager reciprocal relations [1], as shown below. However, it departs from the traditional approach of power conjugate variables, which are generally used in the Package Modelica (exceptions include the rotational, translational, and thermal libraries) [19].

The physical quantities and units are represented using the approach described in [10]. Using that ap-

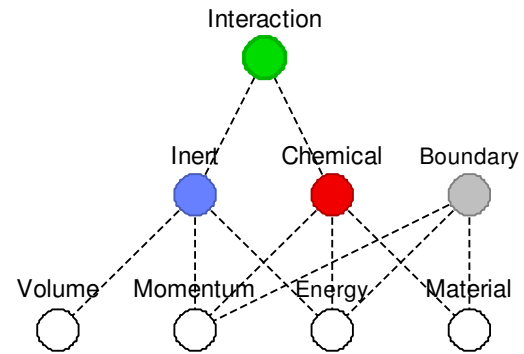


Figure 2: Hierarchy of the connectors

proach, the gas constant and the Faraday constant are both normalized to one. This simplifies the expression of the equations and allows electrons to be described in the same manner as other electrochemical species.

The model contains multiple rectilinear subregions of fixed length (and volume). Each subregion is an instance of the model shown in Figure 5b. Each of the region’s six faces contains a bus connector (expandable). The bus may be populated with a sub-bus for each chemical or electrochemical species present in the region. Optionally, the sub-buses may be first grouped into buses for the mixtures or phases. By default, the length of the vector of momentum connectors is one—representing only normal velocity and force. However, the transverse directions may be included as well; the models adapt accordingly.

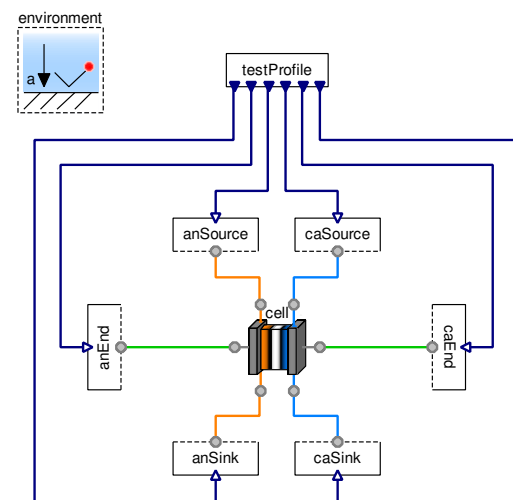


Figure 3: Diagram of a test model that imposes boundary conditions on the cell

In addition to the connectors, the subregion model may contain instances of models to represent species, reactions, and the total volume. A species model describes the advection, diffusion, and storage of material and momenta for a single electrochemical species (e^- , H^+ , H_2 , H_2O , N_2 , or O_2). The species model con-





Within Icon(s)	Name/Quantity	Flow	Effort
	Volume	Volume $V [L^3]$	Pressure per temperature $P/T [NL^{-3}]$
	Linear momentum	Force $m\Phi [MLT^{-2}]$	Velocity per temperature $\phi/T [NTL^{-1}M^{-1}]$
	Energy	Power $\dot{U} [L^2MT^{-3}]$	Reciprocal of temperature $1/T [NT^2L^{-2}M^{-1}]$
	Material	Current $\dot{N} [NT^{-1}]$	Chemical potential per temperature $\mu/T [1]$

Table 1: Summary of connectors used in the models. The dimensions are noted in terms of mass (M), length (L), time (T), and particle number (N). Since the gas constant and the Faraday constant are both normalized to one, charge and thermodynamic temperature are not taken to be independent dimensions.

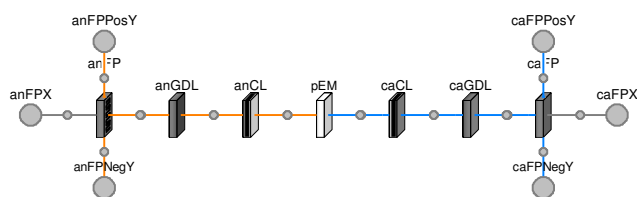


Figure 4: Diagram of a quasi-2D cell

nects to the boundaries and the interaction connector. Optionally, the species models may be nested within a mixture models, in which case the connections to the boundaries are indirect. The species, mixture, and subregion models allow the Cartesian axes to be enabled independently (by parameter), as long as one axis is enabled. As such, the boundary bus connectors of the subregion and mixture models are conditional. The array of boundary connectors in the species model has size $\{2, 1\}$, $\{2, 2\}$, or $\{2, 3\}$, where the first index represents the face (1 or 2) and the second index represents one of the enabled axes.

The species are connected through the expandable interaction connector. In the connection, each species's chemical connector is named by the chemical formula of the species. The inert connector is simply named "inert." In order to prevent nonlinear systems of equations, the connection among species is mathematically direct. Each of the species interacts as if all other species were the same. For instance, each gaseous species interacts equally well with other gaseous species as with the solid. Stated alternatively, the species are "colorblind," which, in the case of volume, is consistent with the basis of Amagat's law [29]. While this is a strong assumption, it can be alleviated by choosing smaller regions, especially where the subregion boundaries are at or near the phase boundaries.

A reaction model exchanges material, momentum, and energy among multiple species. The reaction

models may be used to model chemical or electrochemical reactions. In the chemical case, no material, momentum, or energy is stored. Then, the reaction model simply imposes the stoichiometric constraints (conservation of material), momentum rate balances (without loss), and energy rate balance. Chemical potentials, velocities, and temperatures, are equal in the chemical reaction model. There is no irreversibility; all of the loss is included in the instances of the species model. In the case of an electrochemical reaction, the electrochemical double-layer capacitance is included to account for the electrostatic potential. In the case of the HOR, electrons and protons are stored in equal amounts. Since there is no loss in the reactions models, the net reactions may be partitioned according to convenience, with no mathematical effect. For example, the ORR (Eq. 1b) is modeled as the net PEMFC reaction (Eq. 1c) and the HOR in reverse (Eq. 1a). Since H_2 is not present in the cathode according to the model, it is only an intermediate step without storage and without loss. If additional species are present and interacting (e.g., H_2O_2), they must be included as instances of the species model and joined with the appropriate side-reactions.

At the top level of the subregion, an instance of the volume model is included. It connects to the "inert" sub-connector of the interaction bus to subtract the total volume of the region. Since volume is the flow variable, the partial volumes of the species must sum to the total volume.

Multiple instances of the subregion model are arranged and connected in up to three dimensions to create a region. Figure 5a shows a region, where the subRegions icon represents a 3D array of subregions. The layers of the PEMFC are regions. They are connected as shown in Figure 4 to create the cell model.

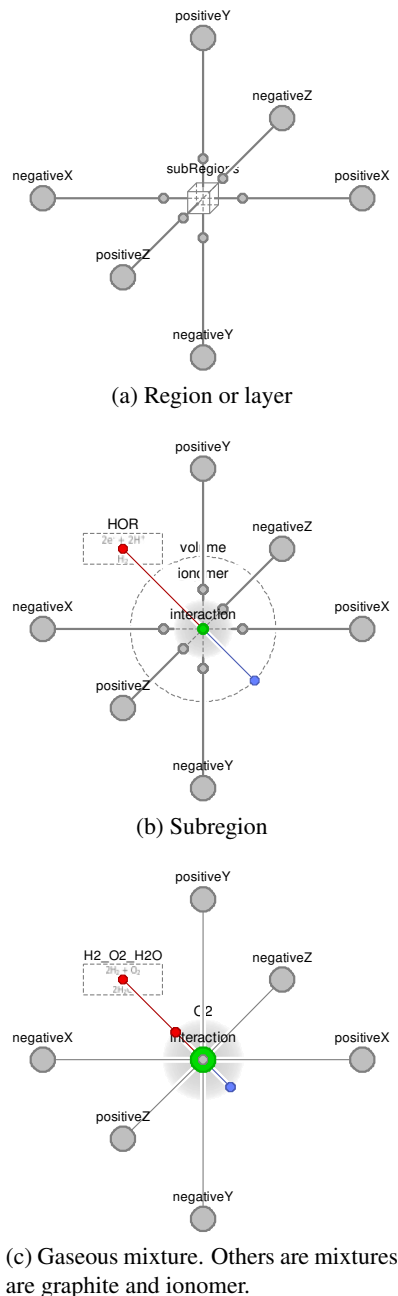


Figure 5: Diagrams of low-level models

At the top/test level of the model, shown in Figure 3, an instance of the cell model is connected to models that impose boundary conditions.

4 Equations

4.1 Physical characteristics

The thermodynamic properties are implemented using the approach of McBride et al. [16], which gives specific heat capacities at constant pressure as seventh-order polynomials of temperature. These are the correlations which are used for ideal gases in the Modelica Media library. The pressure-volume-temperature cor-

relations are implemented using the virial equation of state in the form that is explicit in specific volume [11]. That way, incompressible species and ideal gases can be represented by the same equation with only changes to the constants.

The generalized resistivities for material, momentum, energy, and volume are gathered from a multitude of sources. First, the rigid-sphere assumption may be used from kinetic theory [21]. Second, the correlations of NASA Glenn (formerly Lewis) are implemented where available for the momentum and thermal resistivity (from viscosity and thermal conductivity) [25, 25]. Finally, property tables may be used to set parameters (e.g., [13]). The implementation allows any of these options.

4.2 Species Model

Material is exchanged or transported into port i according to equation 2a, where A_j , L_j and Φ_j are the length and linear momentum along the axis of transport, respectively. The generalized material resistivity is Γ_N . The effective cross-sectional area is the product of the geometric cross-sectional area (A_j) and a factor (k) that accounts for roughness, porosity, tortuosity, and similar properties of the solid structure through which the transport occurs. The parameters β are the Onsager coupling coefficients. By Onsager reciprocal theory, the coefficients β_{ij} equals β_{ji} , where i and j are indexes to the quantities selected according to the theory [1]. The other variables in the equation are efforts and flows from Table 1.

$$\frac{L_j}{kA_j} \Gamma_N \left[\dot{N}_i + \beta_{NV} \left(\frac{P_i}{T_i} - \frac{P}{T} \right) - \beta_{N\Phi} \left(\frac{\phi_i}{T_i} - \frac{\phi}{T} \right) + \beta_{NU} \left(\frac{1}{T_i} - \frac{1}{T} \right) \right] = \left(\frac{\mu_i}{T_i} - \frac{\mu}{T} \right) \left(1 + e^{\pm \Phi_j \Gamma_N / kA_j} \right) \quad (2a)$$

$$\frac{L_j}{kA_j} \frac{\Gamma_V P}{vT} \left[\frac{\delta V}{\delta t} - \beta_{VN} \left(\frac{\mu_i}{T_i} - \frac{\mu}{T} \right) - \beta_{V\Phi} \left(\frac{\phi_i}{T_i} - \frac{\phi}{T} \right) + \beta_{VU} \left(\frac{1}{T_i} - \frac{1}{T} \right) \right] = - \left(\frac{P_i}{T_i} - \frac{P}{T} \right) \left(1 + e^{\pm \Phi_j \Gamma_V / kA_j} \right) \quad (2b)$$

$$\frac{L_j}{kA_j} \frac{\Gamma_{\Phi_j}}{mT} \left[m\dot{\Phi}_i + \beta_{\Phi V} \left(\frac{P_i}{T_i} - \frac{P}{T} \right) - \beta_{\Phi N} \left(\frac{\mu_i}{T_i} - \frac{\mu}{T} \right) + \beta_{\Phi U} \left(\frac{1}{T_i} - \frac{1}{T} \right) \right] = \left(\frac{\phi_i}{T_i} - \frac{\phi}{T} \right) \left(1 + e^{\pm \Phi_j \Gamma_{\Phi_j} / kA_j} \right) \quad (2c)$$

$$\frac{L_j}{kA_j} \frac{\Gamma_U}{T^2} \left[\dot{U}_i + \beta_{UV} \left(\frac{P_i}{T_i} - \frac{P}{T} \right) - \beta_{U\Phi} \left(\frac{\phi_i}{T_i} - \frac{\phi}{T} \right) - \beta_{UN} \left(\frac{\mu_i}{T_i} - \frac{\mu}{T} \right) \right] = - \left(\frac{1}{T_i} - \frac{1}{T} \right) \left(1 + e^{\pm \Phi_j \Gamma_U / kA_j} \right) \quad (2d)$$

$$\frac{\delta N}{\delta t} = \sum \dot{N}_i \quad (3a)$$

$$\frac{\delta m\Phi}{\delta t} + mNa = \sum m\dot{\Phi}_i \quad (3b)$$

$$\underbrace{c_P \frac{\delta T}{\delta t}}_{\text{thermal}} + \underbrace{P \frac{\delta V}{\delta t}}_{\text{volumetric}} - \underbrace{\phi \frac{\delta m\Phi}{\delta t}}_{\text{mechanical}} - \underbrace{\mu \frac{\delta N}{\delta t}}_{\text{electrochem.}} = T \left(\underbrace{\frac{1}{T_i} \dot{U}_i}_{\text{thermal}} + \underbrace{\frac{P_i}{T_i} \frac{\delta V}{\delta t}}_{\text{volumetric}} - \underbrace{\frac{\phi_i}{T_i} m\dot{\Phi}_i}_{\text{mechanical}} - \underbrace{\frac{\mu_i}{T_i} \dot{N}_i}_{\text{electrochem.}} \right) \quad (3c)$$

The factors of the form $(1 + \exp(\pm \Phi_j \Gamma / kA_j))$ account for mixed advection and diffusion. The argument to the exponential is the Péclet number (Pe), which is ratio of advective to diffusive flow. In the case that the advective flow is in the positive direction, the argument will be negative for the negative-facing boundary and positive for the positive-facing boundary. The factor can be interpreted as adjusting the length of diffusive transport according to the extent of drift current or bulk velocity. In the case that there is no bulk velocity, the factor is two; the length from the center of the region to the port is half of the length of the subregion along that axis. Under isothermal conditions, the equations reduce to Fick's law (in the case of chemical species) and Ohm's law (in the case of electrons or holes). In the case that $Pe \rightarrow \infty$, the effort at the positive-facing boundary is equal to the value in the bulk of the region. That is, properties are propagated in the downstream direction. Meanwhile, the relationship between the effort of the negative-facing boundary is related to the effort in the bulk of the region by pure diffusion with the full length of the subregion. The relationships reverse when advective flow is in the opposite direction.

The exchange and transport equations for volume, momentum, and energy are similar to that for material.

If there is only diffusion, then the transport equations for transverse momentum, if included, reduce to the case of Couette flow. The transport equation for energy reduces to thermal conduction when there is no advection and the other efforts are uniform. It is differentially equivalent to Fourier's law. Otherwise, the case is thermal convection—combined advection and diffusion.

In the case of exchange rather than transport, the A/L factor is combined as characteristic length (L^*). It must be calibrated by parameter identification or determined empirically.

The exchange equation for momentum, like the Stefan-Maxwell equation, describes the drag forces between species traveling at different velocities through a mixture [2, p. 538]. However, this approach more manageable. As stated by Cussler, the "Stefan-Maxwell equation is almost never used because it is difficult to solve mathematically, even in the simplest cases" [6].

The Onsager formulation allows gradient of one type of effort to affect the flow rate of quantities besides its conjugate pair. Advection is described in this manner; the same gradient that drives material flow

also drives other flows.ⁱ The difference in velocity normal to the face is coupled to the material flow rate in a reciprocal manner as the difference in chemical potential is coupled to the momentum flow rate.

The exchange/transport equations allow there to be storage within the region, even due to transport along a single axis, because the rates into two faces is not necessarily equal and opposite. In the case of pure diffusion, the rate of intake is proportionally to the second gradient of the effort. The rate balance or conservation equations are given by Equation 3. Einstein notation is used in the summations of the energy rate balance. The form of the energy equation follows from the Onsager approach [1].

5 Discussion

The exchange and transport parameters are cast in terms of resistivity instead of conductivity so that index reduction may be initiated by setting the resistance(s) to zero as final. A typical assumption is that all species (at least within a mixture) are at the same temperature. In addition, the total pressures of the species are expected to be the same after a very short time. If liquid water is added, it may be appropriate to assume that it is in equilibrium with the water vapor. With these assumptions, the number of degrees of freedom reduces to that given by Gibbs' phase rule. It states that the number of thermodynamic degrees of freedom is equal to two plus the number of species minus the number of phases [20], [1, pp. 24–49]. In the case of the assumptions that have been mentioned, the natural thermodynamic state variables would be temperature (the same for all species) and the particle numbers of each chemically independent species or group of species in phasic equilibrium. In the model, index reduction generally introduces nonlinear equations and there is a performance tradeoff between fewer states and fewer nonlinear equations.

6 Conclusion

The architecture and equations for the PEMFC model library have been described at a high level. The implementation is modular and flexible. The same approach would support other electrochemical devices such as batteries. The library is being refined and tested. Results will be given and discussed in a future publication

after validation and calibration.

The library is being made available as open source and should appear on the Modelica website (www.modelica.org). Collaboration would be welcomed.

ⁱThe factor which includes the exponential only amplifies or attenuates the effect of an effort gradient on its own conjugate.

Nomenclature

Symbols

A	Area [L^2]
c	Specific heat capacity [1]
U	Energy [$L^2 M T^{-2}$]
k	Area factor [1]
L	Length [L]
m	Specific mass [1]
$m\Phi$	Linear momentum [$L M T^{-1}$]
N	Particle number [N]
P	Pressure [$M L^{-1} T^{-2}$]
Pe	Peclet number [1]
T	Temperature [$L^2 M N^{-1} T^{-2}$]
t	Time [T]
V	Volume [L^3]
v	Specific volume [$L^3 N^{-1}$]
\mathbf{a}	Global acceleration [$L T^{-2}$]
β	Onsager coupling coefficient [misc.]
Γ	Generalized resistivity [$L T N^{-1}$]
μ	Chemical potential [$L^2 M N^{-1} T^{-2}$]
Φ	Particle number times velocity [$L N T^{-1}$]
ϕ	Linear velocity [$L T^{-1}$]

Accents

· Flow rate of $_$ [$\times T^{-1}$]

Superscripts

* Effective or characteristic $_$

Subscripts

i	$_$ of index i
j	$_$ of index j
N	$_$ of material
P	$_$ at constant pressure
V	$_$ of volume
Φ	$_$ of linear momentum
U	$_$ of energy

Acknowledgments

The authors wish to acknowledge support from the Presidential Fellowship of the Georgia Institute of Technology and the Robert G. Shackelford Fellowship of the Georgia Tech Research Institute.

References

- [1] A. Bejan. *Advanced Engineering Thermodynamics*. John Wiley & Sons, 3rd edition, 2006.
- [2] R. B. Bird, W. E. Stewart, and E. N. Lightfoot. *Transport Phenomena*. John Wiley & Sons, 2nd edition, 2002.
- [3] W. Borutzky. *Bond Graph Modelling of Engineering Systems: Theory, Applications and Software Support*. Springer, 2011.
- [4] K. A. Burke. Unitized regenerative fuel cell system development. NASA report TM—2003-212739, Glenn Research Center, Cleveland, OH, Dec. 2003.
- [5] F. E. Cellier and J. Greifeneder. Modeling chemical reactions in modelica by use of chemo-bonds. In F. Casella, editor, *Proc. 7th Int. Modelica Conf.*, Como, Italy, Sep. 2009. Modelica Assoc., Linköping University Electronic Press.
- [6] E. L. Cussler. *Diffusion: Mass Transfer in Fluid Systems*. Cambridge University Press, 2nd edition, 1997.
- [7] K. L. Davies and R. M. Moore. Object-oriented fuel cell model library. *Electrochem. Soc. T.*, 11(1):797–808, Oct. 2007.
- [8] K. L. Davies and R. M. Moore. PEMFCSim: A fuel cell model library in Modelica. In *31st Fuel Cell Seminar & Exposition*, San Antonio, TX, Oct. 2007.
- [9] K. L. Davies, R. M. Moore, and G. Bender. Model library of polymer electrolyte membrane fuel cells for system hardware and control design. In F. Casella, editor, *Proc. 7th Int. Modelica Conf.*, Como, Italy, Sep. 2009. Modelica Assoc., Linköping University Electronic Press.
- [10] K. L. Davies and C. J. Paredis. Natural unit representation in Modelica. In *Proc. 9th Int. Modelica Conf.*, Munich, Germany, Sep. 2012 (submitted). Modelica Assoc.
- [11] J. H. Dymond, K. N. Marsh, R. C. Wilhoit, and K. C. Wong. *Virial Coefficients of Pure Gases. Numerical Data and Functional Relationships in Science and Technology*. Springer-Verlag, 2002.
- [12] Dynasim AB. Dymola: Dynamic Modeling Laboratory, Mar. 2010. Ver. 7.4.

- [13] F. P. Incropera and D. P. DeWitt. *Fundamentals of Heat and Mass Transfer*. John Wiley & Sons, 5th edition, 2002.
- [14] J. Larminie and A. Dicks. *Fuel Cell Systems Explained*. John Wiley & Sons, 2nd edition, 2003.
- [15] C. Mattiussi. The finite volume, finite element, and finite difference methods as numerical methods for physical field problems. volume 113 of *Advances in Imaging and Electron Physics*, pages 1–146. Elsevier Academic Press, 2000.
- [16] B. J. McBride, M. J. Zehe, and S. Gordon. NASA Glenn coefficients for calculating thermodynamic properties of individual species. NASA report TP—2002-211556, Glenn Research Center, Cleveland, OH, Sep. 2002.
- [17] B. A. McCain, A. G. Stefanopoulou, and K. R. Butts. A study toward minimum spatial discretization of a fuel cell dynamics model. In *Proc. Int. Mech. Eng. Congr. Exposition (IMECE2006)*, number IMECE2006-14509, Chicago, IL, Nov. 2006. ASME.
- [18] D. A. McKay, W. T. Ott, and A. G. Stefanopoulou. Modeling, parameter identification, and validation of water dynamics for a fuel cell stack. In *Conf. on Fuel Cell Science, Engineering and Technology*, Orlando, FL, Nov. 2005. ASME. FUELCELL2005-81484.
- [19] Modelica Association. Modelica Standard Library. <http://www.modelica.org/libraries/Modelica>, Dec. 2009. Ver. 3.1.
- [20] M. J. Moran and H. N. Shapiro. *Fundamentals of Engineering Thermodynamics*. John Wiley & Sons, 6th edition, 2008.
- [21] R. D. Present. *Kinetic Theory of Gases*. McGraw-Hill, 1958.
- [22] M. A. Rubio, A. Urquia, L. González, D. Guinea, and S. Dormido. FuelCellLib: A modelica library for modeling of fuel cells. In *Proc. 4th Int. Modelica Conf.*, Hamburg-Harburg, Germany, Mar. 2005. Modelica Association.
- [23] M. A. Rubio, A. Urquia, and S. Dormido. Dynamic modelling of PEM fuel cells using the FuelCellLib Modelica library. *Math. Comp. Model. Dyn.*, 16(3):165–194, Jun. 2010.
- [24] A. Salogni and P. Colonna. Modeling of solid oxide fuel cells for dynamic simulations of integrated systems. *Appl. Therm. Eng.*, 30(5):464–477, 2010.
- [25] R. A. Svehla. Transport coefficients for the nasa lewis chemical equilibrium program. NASA Technical Memorandum NASA, Lewis Research Center, Cleveland, OH, Apr. 1995.
- [26] U.S. Department of Energy. Hydrogen, fuel cells & infrastructure technologies program: Multi-year research, development and demonstration plan. Technical report, Energy Efficiency and Renewable Energy, Oct. 2007. Section 3.4: Fuel Cells.
- [27] N. Wagner, W. Schnurnberger, B. Mueller, and M. Lang. Electrochemical impedance spectra of solid-oxide fuel cells and polymer membrane fuel cells. *Electrochim. Acta*, 43(24):3785–3793, 1998.
- [28] A. Z. Weber, R. M. Darling, and J. S. Newman. Modeling two-phase behavior in PEFCs. *J. Electrochem. Soc.*, 151(10):A1715–A1727, 2004.
- [29] K. W. Woo and S. I. Yeo. Dalton’s Law vs. Amagat’s Law for the mixture of real gases. *SNU J. Educ. Res.*, 5:127–134, 1995.

The Modeling of Energy Flows in Railway Networks using XML-Infrastructure Data

Andreas Heckmann* and Sebastian Streit[◇]
German Aerospace Center (DLR)

* Institute of System Dynamics and Control, Oberpfaffenhofen, D-82234 Wessling

[◇] Institute of Vehicle Concepts, Pfaffenwaldring 38-40, D-70569 Stuttgart

Abstract

This paper introduces a new Modelica package called RailwaySystem Library that provides the capabilities of simulating the energy flow in electrical railway networks on which a fleet of railway vehicles is running. The focus of the library is set upon the interaction of the vehicle and its energy infrastructure, so that energy management aspects may be investigated from a holistic point of view taking the vehicle and the energy supply by the electric power grid into account. However this intention substantially relies on the provision of reliable data of the infrastructure, on the railway network and its power grid. To this purpose the library refers to an open XML-based data format dedicated to railway IT applications. Furthermore, the library is supposed to be used together with arbitrary component libraries to model the energy subsystems such as the vehicle or the power station.

1 Introduction

As public transport in general, railway transport as well has to cope with increasing demands on the reduction of energy consumption and CO₂ emission. This fact motivates activities of the DLR project Next Generation Train [1] regarding energy management in railway vehicles and recently led to the implementation of the Modelica RailwaySystem Library. This package provides the capabilities of simulating the energy flow in electrical railway networks on which railway vehicles are running.

From the modeling point of view two specific problems had to be taken into account. Railway vehicles may be interpreted as energy sources or sinks that are moving in an inhomogeneous network, see e.g. [2]. The network consists of catenaries or third rails that are supplied by power stations and may or may not be separated in isolated sections. Depending on the num-

ber and the instantaneous position and running state of the vehicles different types of flows may occur in parallel: energy may flow from power station to vehicle, or vice versa or from one vehicle to another vehicle.

As a second important aspect, the evaluation of the energy consumption of a vehicle is of course a function of the track characteristics such as length, slope, radius, positions of power station etc. so that data on the infrastructure topology and properties are required [3]. To this purpose, the library provides access to external infrastructure data, that are filed using the railway markup language railML[®]. This is a XML-based data format, advanced by the railML.org initiative [4] and licensed under creative commons conditions (CC By 2.0) [5].

The initial implementation in this paper is dedicated to consider energy consumption due to conduction losses, traction and auxiliary systems such as heating, ventilating and air-conditioning systems in DC urban-railway-networks. cp. e.g. [6]. However the simulation framework of the Railway System library does not introduce any restrictions on the modeling of the energy subsystems and is open for further extensions. In particular, the Railway System Library is supposed to be used together with component libraries such as the AlternativeVehicles [7] or the PowerTrain Library [8].

2 RailML[®] Data Interface

The non-profit railML.org initiative [4] is a consortium of railway companies, software and consulting firms, and academic institutions, that jointly define and advance a common data standard to be used in different railway simulation tools, see e.g. [9]. The addressed fields of applications are rather comprehensive and among others concern operation planning of rolling stock, resource planning of railroads, design of timeta-

bles, event and delay handling. As a consequence, the XML data standard railML[®] contains subschemas for three main areas: infrastructure, timetable, and rolling stock.

Compared to the scope of the railML.org initiative the piece of work to be presented here only covers specific aspects since it is focused on energy consumption. The initial implementation only considers data regarding track topology and geometry. The following section of an railML[®] file that specifies a track section of 2.7 km length is supposed to serve as an illustrative example:

```
<track id="t4">
  <trackTopology>
    <trackBegin pos="0" id="b4">
      <macroscopicNode ocpRef="PS3" />
    </trackBegin>
    <trackEnd pos="2700" id="e4">
      <macroscopicNode ocpRef="PS5"/>
    </trackEnd>
  </trackTopology>
  <trackElements>
    <radiusChanges>
      <radiusChange id="rC4"
        pos="0" radius="900"/>
      <radiusChange id="rC5"
        pos="400" radius="0"/>
    </radiusChanges>
    <gradientChanges>
      <gradientChange id="gC4"
        pos="0" slope="5" />
      <gradientChange id="gC42"
        pos="1000" slope="0"/>
    </gradientChanges>
  </trackElements>
</track>
```

The data set above specifies that the track starts as a curve with 900 m radius which changes to a straight track after 400 m. The gradient at the beginning of the track section is 5 per mill and changes after 1 km to be horizontally aligned. Note that every data element is specified by a XML-file-wide unique identifier *id*, which is required for later referencing that element.

Each Modelica model that wants to access railML[®] data has to contain an instance of the Modelica railML[®] class, see Fig. 1, and has to provide an XML file name as parameter. The railML[®] instance manages an external object that contains a Document Object Model (DOM) [10] tree of the XML data. The railML[®] instance may be addressed by the inner/outer

mechanism so that other model components may easily acquire information from the railML[®] data.

During initialization of the Modelica model the XML-file is read using the XML parser library *expat* [11] published under MIT license [12] together with the wrapper *scew* [13] available under LGPL license [14]. Both open source tools are written in C and therefore may easily be compiled and bound together with translated Modelica code by every Modelica simulation environment such as OpenModelica, Dymola or SimulationX. As well from the legal point of view these two libraries may be distributed with a Modelica library as long as they are delivered as a self-contained code library which is not mixed up with other C-code.

During initialization a railML[®] object is instantiated and the DOM tree is built up by the two parser tools. In addition every track element found in the railML[®] data is assigned to an integer index number and a mapping of each index number to the XML-wide unique track *id* is organized.

3 Specific Modeling Issues

Fig. 1 presents a trivial network in order to give a first impression of the main modeling components of the RailwaySystem Library that are shortly introduced now.

3.1 Connectors

The library defines the following three connectors. The first one is an aggregation of the 3D-mechanical connector *frame* and the electrical connector *pin* and is tailored to connect catenary sections. The following

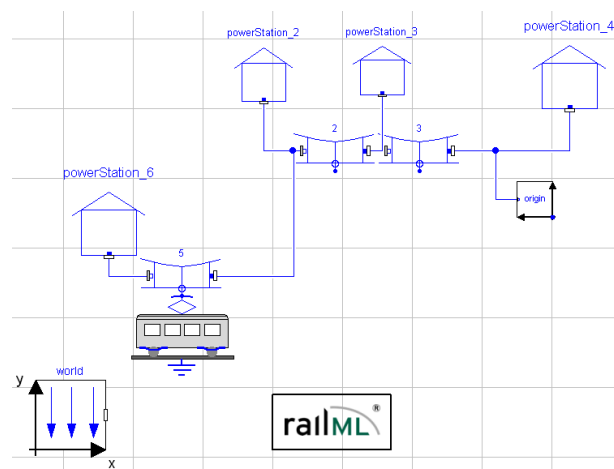


Figure 1: Diagram layer of a trivial network with 3 tracks, 4 power stations and 1 vehicle.

presentation will reveal that the capabilities of the 3D multibody framework are hardly exploited. Neither force or torques balances nor rotations are so far involved in the modeling approach of the library. However future applications may also consider longitudinal dynamics of train sets e.g. during braking or driving up-hill scenarios. In view of such use cases the 3D-mechanical connector *frame* are employed as described below:

```
connector frame_pin
  "supposed to connect catenary
  sections mechanically and
  electrically"
  import Modelica.Mechanics.
    MultiBody.Interfaces.Frame;
  import Modelica.Electrical.
    Analog.Interfaces.Pin;
  Frame frame;
  Pin pin;
end frame_pin;
```

Two other connectors are defined in order to provide the capability of attaching vehicles to the catenary. These connectors only differ in the prefix of the local position variable s , which is an output quantity on the vehicle side, while it is an input variable from the point of view of the catenary.

```
connector slidingContact_a
  "catenary side of catenary-
  pantograph connection"
  extends RailwaySystem.
    Interfaces.frame_pin;
  input Real s "local position"
end slidingContact_a ;
```

```
connector slidingContact_b
  "pantograph side of catenary-
  pantograph connection"
  extends RailwaySystem.
    Interfaces.frame_pin;
  output Real s "local position"
end slidingContact_b ;
```

In particular the definition and the purpose of the variable s is further motivated and explained in the following three sections.

3.2 Vehicle

The *Vehicle* model is a base class and supposed to be extended in order to characterize the energy system of a railway vehicle. The energy system itself may be arbitrary complex and may be modelled using components from the Standard Modelica library together

with components from commercial libraries such as the *AlternativeVehicles* [7] or the *PowerTrain Library* [8].

Important parameters of the *Vehicle* model are *tracksToPass* and *tracksPassOver*:

- The parameter *tracksToPass*, e.g. *tracksToPass*={5,2,3} in Fig. 1, is an integer vector containing the indices of the tracks the vehicle is supposed to run on. The order of the indices corresponds to the sequence of the tracks.
- The real vector *tracksPassOver*, e.g. *tracksPassOver*={0, 3200, 6800, 9800} in Fig. 1, specifies points on the path of the vehicle, at which one track is left and the following is entered.

Important transient variables of the vehicle model are *trackIndicator* and the real quantities s and S :

- The boolean vector *trackIndicator* is of the same length as *tracksToPass*. One and only one element of *trackIndicator* is true, namely the element that is associated to the track the vehicle is currently running on.
- The variable s defines a specific point on the track, the vehicle is currently running on. It is a local, track-specific variable on contrary to S .
- The variable S is a global vehicle-path-specific quantity. In the present implementation S is pre-defined as a function of time, so that motion of the vehicle along its path is preset e.g. as a result of the timetable. Alternatively it is also possible to give the velocity profile as a function of S and evaluate $S = S(t)$ accordingly.

The following table again summarizes the important variables explained above:

parameters	
<i>tracksToPass</i>	track indices
<i>tracksPassOver</i>	specific path points
transient variables	
s	local track position
S	global path position
<i>trackIndicator</i>	boolean track switch

From the purely structural point of view a vehicle instance is connected to all catenary sections that are listed in the parameter vector *tracksToPass* using the

sliding contact connector classes, see Sec. 3.1. However by employing ideal closing switches from the Standard Modelica.Electrical library it is guaranteed that only that electrical connection is closed to which the corresponding value of *trackIndicator* is set to true.

In order to access the railML[®] data and provide information on the e.g. the current gradient, the current values of *s* and *trackIndicator* together with *tracksToPass* are interpreted and passed to appropriate C-functions that extract data from the DOM tree.

3.3 Catenary

The *Catenary* model represents a track segment parametrized with the local length coordinate *s*. Its geometrical and electrical properties such as radius, gradient and electric resistance vary as a function of *s*. The integer parameter *ID* specifies the index to access the RailML database so that the necessary information on the track segment given by the RailML database may be acquired.

A sliding contact connector serves as an interface between vehicle and track and the current value of *s* denotes the current local position of the vehicle. Since the *Vehicle* instance as well as the *Catenary* instance both rely on the value *s*, the definition of the two connectors *slidingContact_a* and *slidingContact_b* in Sec. 3.1 considers the exchange of this variable.

The two other *frame_pin* connectors are supposed to connect different catenary sections. Future versions of the RailwaySystem Library will include the capability to automatically instantiate and connect all track sections found in the railML[®] database, so that the modeling of a complex network structure is substantially facilitated. So far the network structure is to be built up by manually instantiate and connect *Catenary* objects.

Note that variants of the *Catenary* model class are available that consider more than one vehicle running along the same track.

3.4 PowerStation and Origin

The *PowerStation* model is used to introduce transformer substations along the track that serve as voltage supply sources.

The railML[®] data only contains relative information like track lengths specifying the distances to travel from one point to another. In order to be able to set up an at least schematic animation of the traveling vehicles one absolute position has to be defined. This is done by the *Origin* model class.

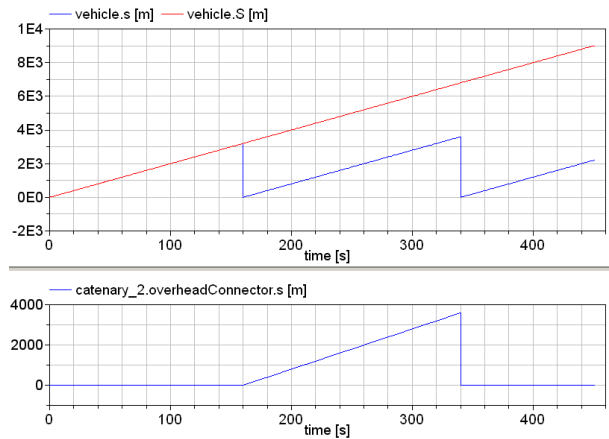


Figure 2: Plot of the variables *s* and *S* as a function of time.

3.5 Exemplary Simulation Sequence

In order to present the general simulation set-up of the RailwaySystem Library the simulation sequence of the trivial network shown in Fig. 1 will now be explained. The considered vehicle parameters are:

- $tracksToPass = \{5, 2, 3\}$,
- $tracksPassOver = \{0, 3200, 6800, 9800\}$,
- $S = 20 \text{ m/s} \cdot t$

where *tracksToPass* and *S* are specified by user input, while the values of *tracksPassOver* are generated by a function that gains information from the railML[®] object during initialization.

According to the upper plot of Fig. 2, the vehicle leaves the first catenary section after 160 s, and the second after 340 s. This corresponds to the length of 3200 m and 3600 m of the first (*ID* = 5) and the second catenary (*ID* = 2) and the constant velocity of 20 m/s. The plot below shows the value of *s* seen from the *catenary_2* point of view. As long as the vehicle is not running on this catenary or track section, respectively, *s* is set to zero.

Fig. 3 demonstrates that the value of the first element of the vector *trackIndicator* is set to true as long as the vehicle is moving along the catenary specified by the first element of *trackToPass*. This applies for the second and the third element of *trackIndicator* accordingly.

The vector *trackIndicator* controls a vector of electrical switches so that the vehicle energy system is linked to that catenary or track section only, the vehicle is currently running on.

In summary, it is the general idea of the simulation set-up that the vehicle instance gathers all information.

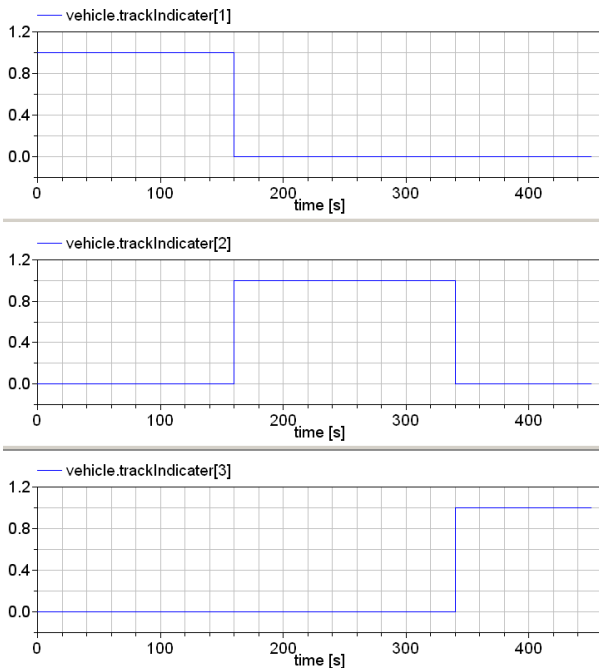


Figure 3: Plot of the boolean variable `vehicle.trackIndicator`.

The vehicle "knows" where, on which track or catenary section it is currently running and it is enabled to access the railML[®] data to acquire infrastructure information accordingly. The vehicle hooks itself up to the current catenary section in order to manage its own energy supply.

4 Application Example

The example model in Fig. 4 presents a small DC-powered urban light-rail network supplied by six power stations where two vehicles are running on six tracks.

Today power stations in DC-powered light-rail networks use rectifiers to provide a load-dependent control of the DC voltage. That means within the valid limits the output voltage is freely adjustable [2]. Therefore it is feasible to model the power stations as constant voltage sources. The nominal voltage used in this case is 750 V which is typical for DC-powered urban light-rail networks.

At a given load the traction current of the railway vehicle depends on the input voltage of the vehicle and thus of the specific position on the track [6]. For this reason the voltage drop alongside of the catenary has to be considered. Taking the resistance load per length into consideration, the voltage drop along the catenary can be calculated according to the length between the

feeding point at the traction substation and the pantograph of the vehicle.

The basis for the calculation of the energy flow within the given network is the simulation of the vehicle trajectory. Based on the available nominal power of the vehicle and the tractive force at starting the tractive force-to-velocity characteristic is calculated. Thus the maximum available traction force can be calculated depending on the actual velocity. Depending on basic parameters of the vehicle e.g. weight, rolling resistance or aerodynamic resistance the driving resistance of the vehicle can also be calculated for each given velocity.

In addition specific parameters of the track resulting from curves, gradients or tunnel (provided via RailML data) lead to additional resisting forces that need to be considered. The movement of the vehicle is simulated by applying all resulting forces to a point mass. From the movement of this mass all necessary data for calculation of the electric network can be derived. The calculated mechanical power is used to derive the electrical power consumption of the vehicle. This power consumption is needed to calculate the traction current during simulation of the electrical network. The vehicle trajectory is performed for a predetermined velocity profile corresponding to the tracks to pass.

To simulate the energy flow within this network the vehicles are modeled as variable current sources using the actual required electric power consumption as input. In this way it is possible to calculate the resulting current sharing based on the electric power re-

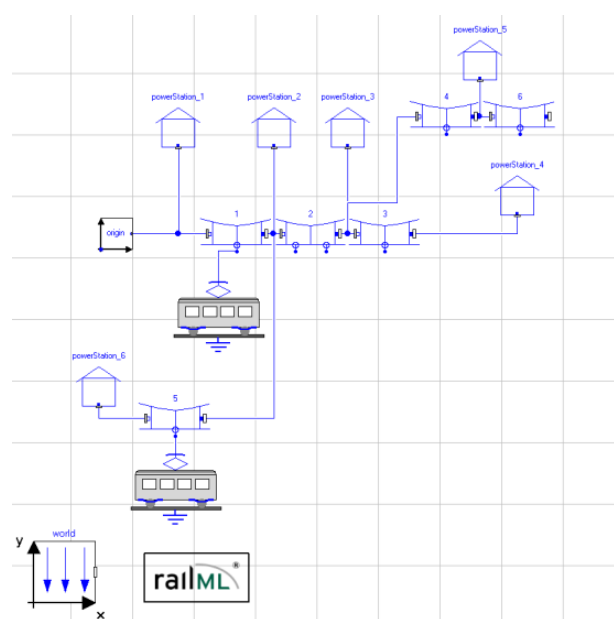


Figure 4: Diagram layer of the network.

quirement of the vehicles, the supply voltage and the voltage drop alongside of the catenaries.

As an example Fig. 5 shows the resulting voltage at the pantographs as well as the traction currents of two vehicles on their pass through section 2 of the exemplary network shown in Fig. 4. The first vehicle (red curves in Fig. 5) enters this section at about $t = 115$ s. It enters the section at a velocity of 40 km/h. Since the scheduled velocity in this section is 80 km/h the vehicle accelerates until it reaches the allowed velocity at about $t = 132$ s. This acceleration is associated to high traction forces resulting in a strongly increasing traction current.

Due to the increasing current the voltage at the pantograph drops during the acceleration to about 700 V. At approximately $t = 183$ s the traction current is again significantly increasing. At this specific point the gradient of the track changes from 0 to 30‰. This gradient abruptly increases the resisting force. To keep the scheduled velocity the traction force has to be increased as well resulting in a higher required mechanical and consequently electrical power consumption of the vehicle.

The traction current remains high until the gradient changes again from 30 to 0‰ at approximately $t = 257$ s. Fig. 5 presents the voltage at the pantograph to jump up to 650 V during this passage.

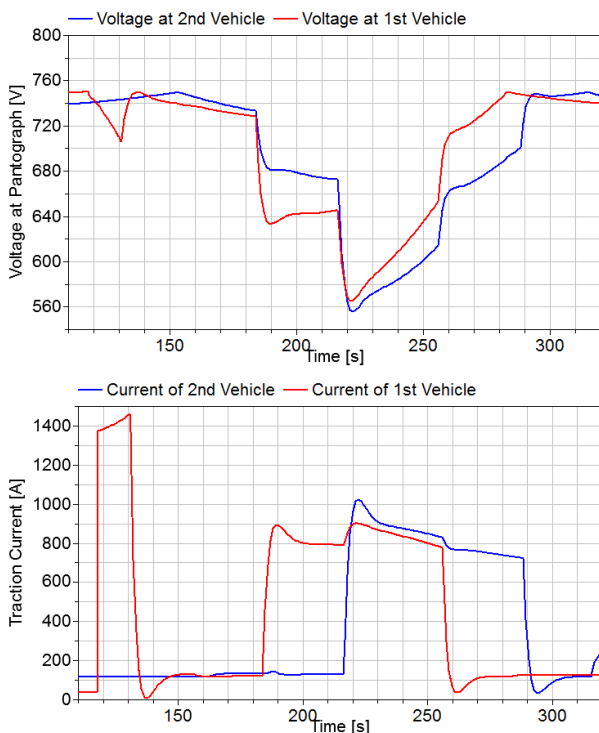


Figure 5: Simulation result of the voltages of both vehicles as a function of time.

The second vehicle (blue curves in Fig. 5) enters this section at about $t = 30$ s later already at a velocity of 80 km/h so that there is no further acceleration needed. When the second vehicle approaches the gradient change its traction current increases for the same reasons as mentioned before. At this point in time both vehicles have a high power consumption that leads to an additional voltage drop in the whole section. The voltage at the pantographs of both vehicles then drops significantly under 600 V which is critical since the minimum permitted voltage in DC-powered light rail networks with a nominal voltage of 750 V is 500 V.

To investigate the influence of energy storage devices as part of an energy management of railway networks a basic model of an electric double layer capacitor a so called Super Cap was also included within the vehicle model.

The Figures 6 and 7 each compare two different scenarios for the usage of these Super Caps. Fig. 6 shows the voltage drop at the pantograph of one vehicle passing the same section as shown in Fig. 5 as well as the state of charge of the Super Cap for two different cases.

Initially the vehicle is at rest and then accelerates up to 80 km/h. It stops at the end of the section. The Fig. 6 demonstrates the influence on the voltage drop during acceleration if state of charge is at 100% at starting time. In the sequence the voltage does not drop until

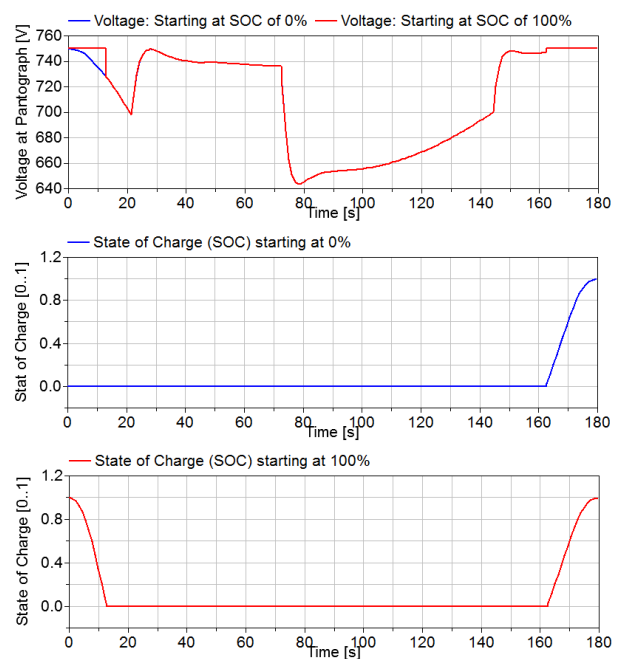


Figure 6: Simulation results presuming two different initial states of the Super Cap.

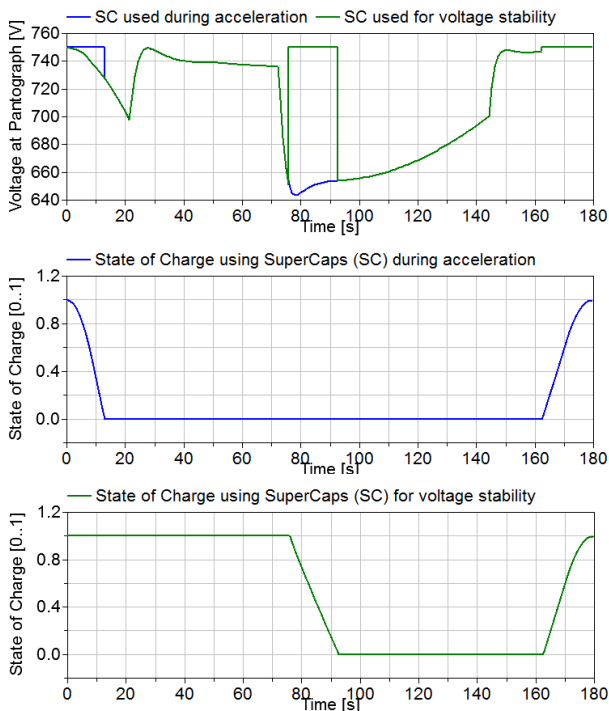


Figure 7: Simulation results associated to two different regimes for the use of Super Caps in operation.

the Super Cap is fully discharged at about $t = 17$ s. Until that point in time the full traction power is provided by the energy storage and no current is flowing between substation and vehicle.

When the vehicle starts braking at the end of the section, the Super Cap is fully charged and the stored energy can be used for the next run. Considering the same vehicle trajectory as in Fig. 6 Fig. 7 delineates another exemplary regime to employ an energy storage device. The Super Cap may not only be used to provide energy for acceleration but may also be exploited in order to ensure voltage stability. Fig. 7 presents the Super Cap not to be discharged until the voltage drops below 650 V. As soon as the voltage drops below this threshold the required electrical power is provided by the energy storage and for about 17 s the traction current is fully supplied through the Super Cap. This way

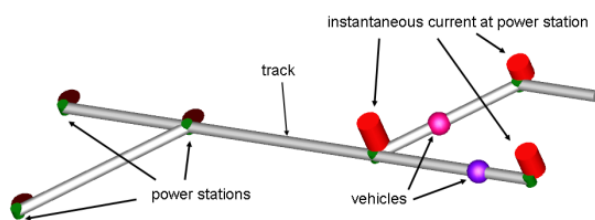


Figure 8: Schematic animation of the simulation.

a temporary voltage drop below critical values can be avoided.

Fig. 8 depicts an animation of the scenario when one vehicle is running along Track 3, while the other is moving between Power Station 3 and 5 at Track 4. The height of the red bars positioned close to each power station illustrate the instantaneous current provided by the associated power station.

The given examples have shown in principle that future investigations of energy flows within complex electric networks including the consideration of energy storage devices can be done using standardized data sets in Modelica.

5 Conclusions and Outlook

In the course of the DLR project Next Generation Train the RailwaySystem Library will be used in order to evaluate the energy reduction potential of an energy management system that takes the vehicle energy system and the power supply infrastructure into account.

Future versions of the RailwaySystem Library will include the capability to automatically instantiate and connect all track sections found in the railML[®] database, so that the modeling of a complex network structure is substantially facilitated.

References

- [1] J. Winter, E. Mittelbach, and J. Schykowski, editors. *RTR Special - Next Generation Train*. Eurailpress, DVV Media Group, 2011.
- [2] H. Biesenack, G. George, G. Hofmann, A. Schmieder, E. Braun, K. Girbert, R.C. Klinge, R. Puschmann, S. Röhlig, E. Schlechter, E. Schneider, A. Stephan, and G. Zimmert. *Energieversorgung elektrischer Bahnen*. Teubner Verlag, Wiesbaden, 2006.
- [3] D. Hürlimann. *Objektorientierte Modellierung von Infrastrukturelementen und Betriebsvorgängen im Eisenbahnwesen*. PhD thesis, ETH Zürich, 2001.
- [4] RailML[®]. <http://www.railml.org/web/>. [Online; accessed 20-February-2012].
- [5] Creative Commons License. http://en.wikipedia.org/wiki/Creative_Commons_license. [Online; accessed 15-May-2012].

- [6] S. Röhlig. *Beschreibung und Berechnung der Bahnbelastung von Gleichstrom-Nahverkehrsbahnen*. PhD thesis, TU Dresden, 1992.
- [7] Th. Braig, H. Dittus, J. Ungethüm, and T. Engelhardt. The Modelica library AlternativeVehicles for vehicle system simulation. In *21. Symposium Simulationstechnik, SIM 2011*, Winterthur, Suisse, Sept., 7. - 9. 2011.
- [8] J. Tobolář, M. Otter, and T. Bunte. Modelling of Vehicle Powertrains with the Modelica PowerTrain Library. In *Systemanalyse in der Kfz-Antriebstechnik IV*, pages 204–216. expert Verlag, 2007.
- [9] A. Nash and D. Hürlimann. Railroad simulation using OpenTrack. In J. Allan, C.A. Brebbia, R.J. Hill, G. Sciutto, and S. Sone, editors, *Computers in Railways IX*. WIT Press, 2004.
- [10] Document Object Model (DOM). http://en.wikipedia.org/wiki/Document_object_model. [Online; accessed 15-May-2012].
- [11] expat. The expat XML parser. expat.sourceforge.net/. [Online; accessed 16-March-2012].
- [12] MIT License. http://en.wikipedia.org/wiki/MIT_License. [Online; accessed 15-May-2012].
- [13] scew. Simple C expat wrapper. www.nongnu.org/scew/. [Online; accessed 16-March-2012].
- [14] LGPL License. http://en.wikipedia.org/wiki/LGPL_license. [Online; accessed 15-May-2012].

Implementation of a Modelica Library for Energy Management based on Economic Models

Dirk Zimmer, Daniel Schlabe
Deutsches Zentrum für Luft- und Raumfahrt.
Münchner Strasse 20, 82234 Weßling, Deutschland
{dirk.zimmer, daniel.schlabe}@dlr.de

Abstract

The use of modeling paradigms for physical systems can in some instances be stretched to reach other domains. This paper presents one such example: it describes the design of a Modelica library that implements economic models to be used for the purpose of energy management. The design principles of this library such as the use of pseudo-physical connectors are outlined and examples for managing energy sources and loads are discussed.

Keywords: Energy management, Load management, Economic models, Object-oriented modeling.

1 Introduction

This paper presents the modeling of energy management tasks by the use of economic models. In this approach, each provider of energy and each consumer is characterized by a specific cost function. A global market or a set of local markets then decide about the distribution of energy flow.

To this end, a new Modelica library has been developed. It supports the modeler in the design of his or her energy distribution system and derives an (at least partly) optimal solution for the distribution based on the provided cost functions.

The library is not coupled to any specific physical domain. All its components concern energy in its most abstract form. In fact, many energy management tasks involve multiple physical domains and therefore a domain-specific approach would be of limited value.

The library is currently split into two sub-libraries that are geared towards different application domains: source management and load management. It is still under development and currently not publicly available.

2 Economic Models for Energy Management

2.1 State of the Art

The links between models and theories used in micro-economics and typical tasks of an energy management function are very close. In both cases, there is a set of providers and a set of consumers. The consumers pay a price of a utility depending on the availability or production of the providers. The main difference is the type of the utilities. In micro-economics this is typical any kind of product, for an energy management the utility is power or energy.

The application of economic models for a power management is already demonstrated in [9]. An energy manager based on economic models for the electrical system of automobiles, especially for hybrid cars, has been studied in [1] and [6]. Additionally, available methods for energy management of aircraft electrical systems can be found in [8].

The main idea behind this market-oriented approach is the usage of power p over price v functions for each source/provider and consumer/load as illustrated in Figure 1.

$$p = f(v)$$

These functions describe how much price a load is able to pay for a dedicated power and how much power a source will provide for a certain price respectively. These functions could be determined by e.g. the efficiency or the priority of a component.

Since p denotes the outflowing power, the cost functions are typically positive for sources and negative for loads.

Subsequently for all sources and loads the sum-functions are calculated as shown in Figure 2. The intersection of load and source sum-functions determines the current price and thus the power of each component.

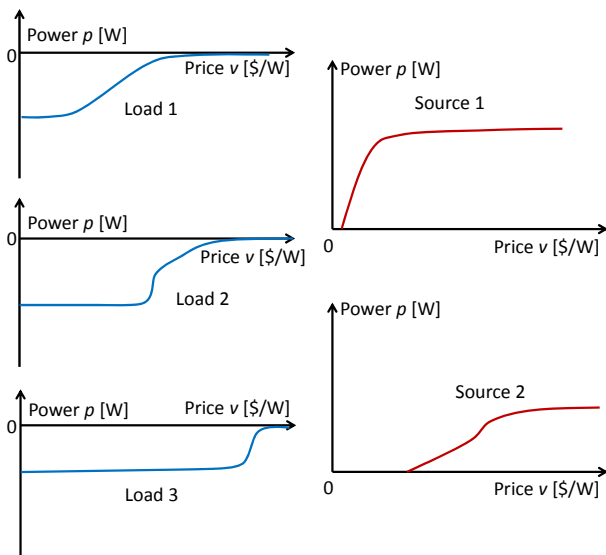


Figure 1: Cost-functions of single loads and sources.

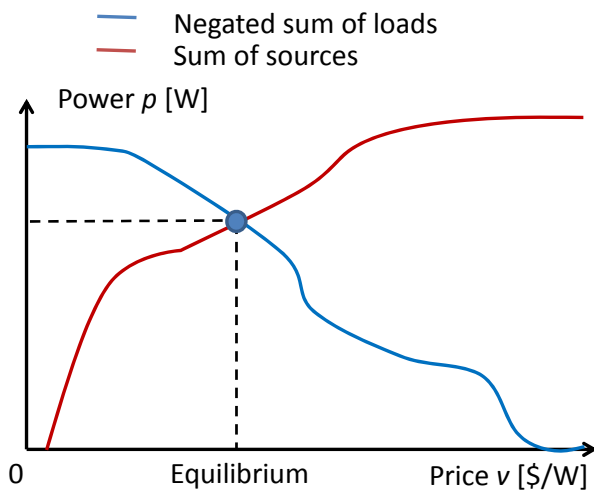


Figure 2: Sum-functions and equilibrium.

The advantage of such an approach is the integration of different relevant aspects like efficiency of the sources or availability of the consumers for an energy manager in one single characteristic cost function. Furthermore, this enables the modeling of sources and consumers in an object-oriented way and thus an easy set-up of an energy management function of a dedicated system within an early stage of design.

2.2 Limitations

To guarantee the existence of a unique intersection of load and source cost-functions, these have to be monotone and continuous. If this restriction is not maintained, one has to guarantee with other means that a stable intersection can be found in either case.

In addition, economic models are best suited for finding an optimal solution at one specific time in-

stant, but not for optimizing the energy consumption predictively regarding dynamic influences. For this case, further means are needed that have to be integrated to these models.

2.3 Scientific Contributions of this paper

Based on the described state of the art, this paper demonstrates the implementation of a market-oriented energy management library in Modelica. Therefore the library including its components and the working principles are outlined in the following sections.

New concepts for dealing with non-monotone cost functions of sources are introduced. For this task, several rounds of negotiation are being used. Multiple negotiation rounds are also used for dealing with switchable and continuous loads in one system to reach a maximum availability of loads.

The modeling of energy systems is not confined to models for sources and loads. Hence also further components like limiters or transformers are considered that modify the cost-functions in a dedicated way.

3 Fundamental design of the library

The goal of this paper is to describe how such economical models for energy management can be modeled in a truly object-oriented way. The idea is that energy distribution systems can be assembled from basic components such as producers and consumers. Also the modeler shall not be directly concerned with the cost functions. Instead the cost-functions should be derived by parameters such as efficiency or priority levels.

To this end, a Modelica library has been developed. In this section, we present its common interface and the most basic components.

3.1 Connector design

The connector of the energy management library is a so-called pseudo physical connector. This means that it mimics the characteristics of classic physical connectors without describing actual physical quantities. In concrete terms: the connector contains a pair of a potential variable and a flow variable just like a physical connector. In this way, we profit from the advanced support of physical connectors (like the check of balanced models) in Modelica.

The potential variable of the connector is the price per watt [\$/W] and the flow variable is the power outflow [W]. A positive value for the power

outflow is typical for a source. Consequently consumers have negative values of their flow variable. Similar pairs have already been suggested during the 1970s in [2] and [3] and enable a more natural modeling than sheer System Dynamics for Modelica [5].

The product of the potential variable and the flow results in the amount of money that is transmitted through the connector (negative values represent costs, positive values represent income). The money is of course virtual and not related to any real currency.

A connection between a set of connectors thus represents an ideal market where all participants pay or receive an equivalent price for an equivalent product.

Listing 1: Code of the power socket.

```
connector Socket

  parameter Integer n=1;

  PricePerWatt price[n];
  flow SI.Power power[n];

end Socket;
```

Listing 1 presents the Modelica code of the connector. Evidently, price and power represent not scalars but vectors of a parameterized size n . The reason for this is explained in section 4.6. For the moment, let us continue by pretending these are scalars. We simply assume: $n=1$.

3.2 Icons





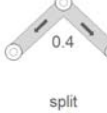
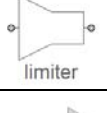

A component of the library may represent a source of energy, a consumer, a transformer of energy or redistributors.

These are all components that also occur in many physical domains such as electric systems. However, since this library shall be domain independent, no symbols of such libraries shall be used.

There are only a few domain neutral symbol languages. One of them is bond graphs. For our purpose bond graphs [4] are however too low-level and too technical. For instance there is no distinction between a source and a sink in bond graphs.

Another set of icons has been developed for the Energy Systems Language developed in the field of ecology by Howard T. Odum [7]. It is also not directly usable for our purpose, but at least the abstract forms used in this language inspired the design of our set of symbols that is listed in Table 1.

Table 1: Icons used for energy management.

	Source / Producer
	Sink / Consumer / Load
	Waste
	Transformer
	Split
	Limiter
	One-way

A source can represent a source of fuel or an energy producer such as a power plant. The sink is its counterpart element. It mostly represents a consumer. The waste element is a special case for the sink that enables the system to waste energy.

Energy can be transformed into other forms by imposing further costs using a transformer. The split element can be used to distribute energy into different branches. For instance in a combined heat and power plant 40% of the power is electricity and the remaining 60% are available as heating power.

The components one-way and limiter are explained in section 4.4 and section 5.2 respectively.

3.3 Example

Given the set of components, it is now possible to compose an energy distribution system. Figure 3 shows the model diagram of an example system. Here, two sources are available: one for heating and one for electricity. Two consumers model the respective demand. In addition there is the possibility to use electricity for heating. A waste element ensures that energy can be dumped in the unlikely case that the electricity demand may fall below the idle power output of the electricity generation plant.

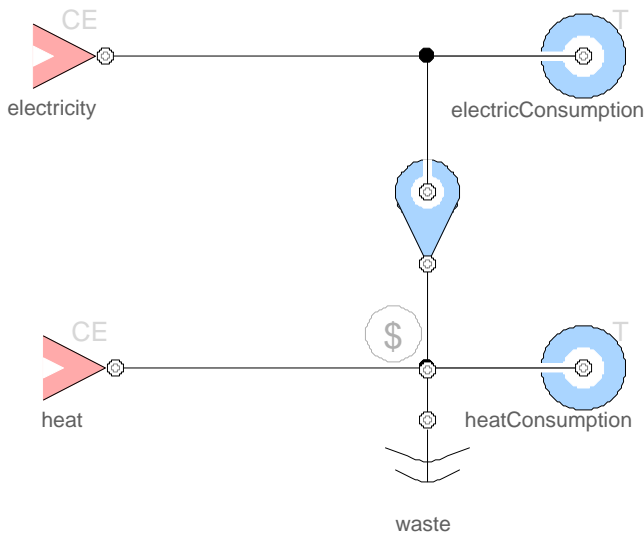


Figure 3: The model diagram of an example market.

3.4 Solving the non-linear systems of equations

All component models contain a description of their cost function that expresses the price as function of the power. The connection of this components leads then to (typically) non-linear equation systems. If all cost functions are strictly monotonic increasing or decreasing, there will be a unique solution.

Depending on the cost-function and the specific connection structure, a simulation software such as Dymola might be able to solve this non-linear equation system, but in some practical examples this turns out not to be the case.

Hence we have developed an auxiliary controller unit that regulates the price v on the market by a simple differential equation. The controller may compensate for any lack or excess of power p . It increases the market price in case of a power outflow ($p > 0$) due to a lack of power and decreases the price in case of a power inflow ($p < 0$) due to excess of power.

$$dv/dt \cdot T = p$$

where T is an arbitrary time constant.

This controller is typically applied to a connection set. In the diagram of Figure 3, it is depicted as grey "\$" placed in a circle. With this element, it is possible to find the solution in robust way by approaching steady state. The drawback of such a controller is that it makes the system potentially stiff and requires implicit solvers such as DASSL for the efficient simulation of the system.

The application of such a controller could probably be avoided if there exists a Modelica language construct to suggest suitable tearing variables.

4 Application Domain: Source Management.

4.1 Motivation

In this application domain, we want to fulfill a given consumer demand by using the most efficient combination of sources available. Hence the cost functions take into account the efficiency of sources and subsequent processes of energy transformation.

4.2 Derivation of cost functions

In this scenario, the consumer demand is regarded as a given that is required to be fulfilled at any cost. Hence modeling the cost function of a consumer is very simple: A consumer is the equivalent to an ideal flow sink. Prescribing the flow variable for any potential price per watt while leaving the price to be determined by other parts of the system:

$$p = -\text{demand}$$

The waste element is a special case of a consumer. An ideal waste element is similar to an ideal diode. It is a sink of zero flow for prices above zero and consumes arbitrary amounts of energy at a price of zero. A price below zero means that the producers would have to pay for their energy to be consumed. Although this actually occurs in real markets, the waste element can be used to prevent such cases.

$$s = \text{if } (s > 0) \text{ then } p \text{ else } v;$$

$$0 = \text{if } (s > 0) \text{ then } v \text{ else } p;$$

where s is a curve parameter

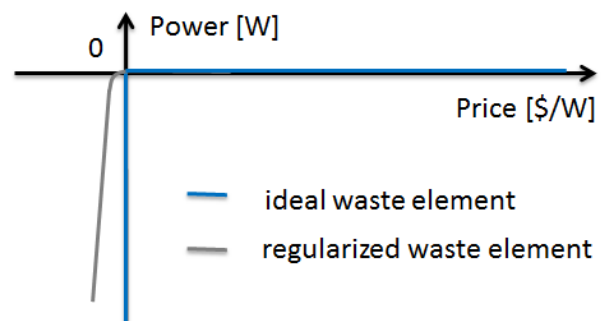


Figure 4: Cost function of a waste element.

Modeling sources is a little more difficult. The price shall reflect the efficiency of energy use. The simplest case is a source of constant efficiency. In the ideal case, this source stipulates the price for any

arbitrary power output to be the inverse of the efficiency:

$$v = 1/\text{efficiency}$$

No real source of energy is unbounded. All sources have a maximum capacity and many of them have an idle power output beyond which their production cannot decrease. These limitations can be modeled by a step function.

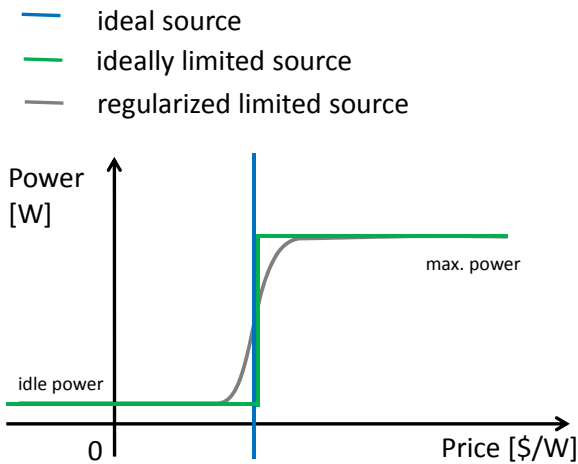


Figure 5: Cost function of different source models.

Finally, a split element can be used to model the separation of power into distinct branches by a fixed fraction. It distributes the power inflow p_{in} into two power outflows p_{out1} and p_{out2} by a given fraction R . The split element is connected to markets with a different price per watt. The price per watt of the power inflow v_{in} is then the weighted mean of the two outflow prices: v_{out1} and v_{out2} . Here are the corresponding equations to relate the three connectors:

$$\begin{aligned} p_{in} + p_{out1} + p_{out2} &= 0; \\ p_{out1} \cdot (1-R) &= p_{out2} \cdot R; \\ v_{in} &= v_{out1} \cdot R + v_{out2} \cdot (1-R); \end{aligned}$$

4.3 Regularizing the cost functions

For the numerical solution, it is advantageous if all cost functions are continuous and strictly monotonic functions. Then a unique solution is guaranteed in case the total demand can be met. But the curves for the ideal limiter or the ideal waste element substantially differ from this requirement. They represent multi-valued functions that are also strictly monotonic increasing or decreasing. Indeed their modeling would require the use of parametric curves such as for ideal diodes. To avoid this effort and the resulting

numerical problems, a regularization scheme is applied.

The regularization is indicated by the grey curves in Figure 4 and Figure 5. For its realization, a mixture of sigmoid and exponential functions is used. The precise realization is somewhat arbitrary and also of no particular importance and hence has been omitted here.

The regularization is of course a further potential cause of stiffness and/or implies a loss of precision. The trade-off between precision and stiffness can be set by fudge parameters. These are provided globally by an outer model so they do not have to be set of each element individually.

4.4 Example 1: A combined power generator

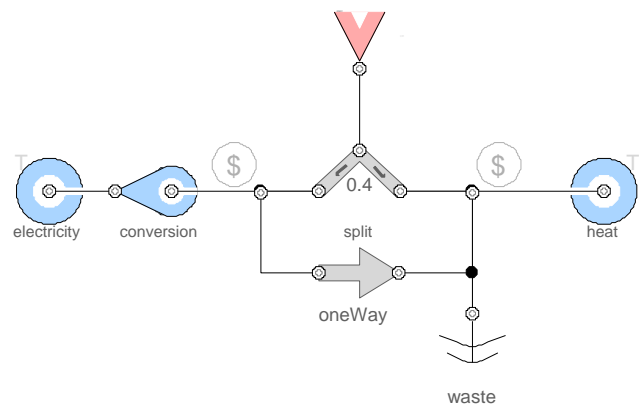


Figure 6: Model diagram of a combined power generator and two corresponding consumers for electricity and thermal energy (heat).

Figure 6 presents the example of a combined power generator of electricity and heat. Up to 60% of the thermal energy can be converted into electricity. This is modeled by a combination of a split element and a one-way component that acts like a diode: power can only flow in one direction.

The loss in conversion between thermal and electric energy is modeled by a transformer component. Both consumer models stipulate the total power demand that is varying over time.

For the simulation, the electric consumption is constantly decreasing from 250 kW to 100 kW. The demand of thermal energy is constantly increasing from 50 kW to 500 kW. The impact on the price can be observed in Figure 7. It contains the simulation result for the price per Watt for both consumers.

Due to the initial high demand for electricity, the consumers of thermal energy do not have to pay anything at all (the price is actually even slightly below zero because of the regularization of the waste element). The generation of electric energy produces sufficient heat as side product.

During the simulation, the demand shifts towards the need for thermal energy. Then the bill needs to be split. Electric energy still remains more expensive than thermal energy because it needs to be converted (at loss) from thermal energy and the combined producer can control how much of that needs to be converted.

This example demonstrates how the cost-function of a more complex source like a combined generator can be modeled in a true object-oriented way by combining simple components.

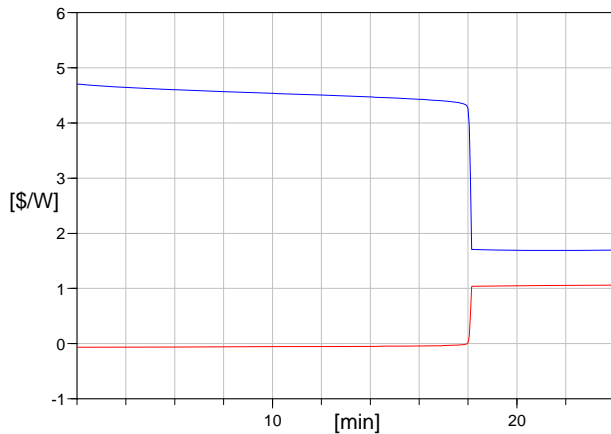


Figure 7: Price development of thermal energy (red) and electric energy (blue).

4.5 Treatment of non-monotonic cost-functions

The presumption that the cost function is strictly monotonic increasing is not realistic for a large set of power generators. Many of them have an ideal operating range that does not start at idle power. This means that when these generators are used for low power output they can be very inefficient. The multi-valued cost-function of Figure 8 represents such a characteristic curve.

The solution of systems with such cost functions can be numerically very difficult and often there are multiple equilibriums in the market. Finding the optimal equilibrium is a very demanding optimization problem that in general cannot be handled in polynomial time. Hence a robust handling of such non-monotonic cost function requires a good solution strategy.



Figure 8: A non-monotonic, multi-valued cost function (red) and a corresponding monotonic, single-valued hull curve (grey).

In this paper, we propose a bullying strategy. It reflects a behavior that also exists in real markets. Big players, in our case large and potentially very efficient power generators, compete for a contract. They pretend to be more efficient than they actually are. When the order finally turns out to be too small to be efficiently handled by the big player, the contracts are handed over to small players by issuing sub-contracts. The final point of equilibrium is hence determined in several rounds of negotiation: first the big players then the smaller players.

In our library such a bullying strategy is implemented by creating hull curves in multiple rounds of negotiation. Figure 8 shows the effective cost-functions for our producer. However, in the first round of negotiation this curve is not used but the grey hull curve instead.

The hull curve must be monotonic increasing and must always be greater or equal than the effective cost curve. Within these constraints, it should be as low-valued as possible. In those sections where the hull curve does not coincide with the effective cost curve, the producer is hence pretending to be more efficient than he actually is.

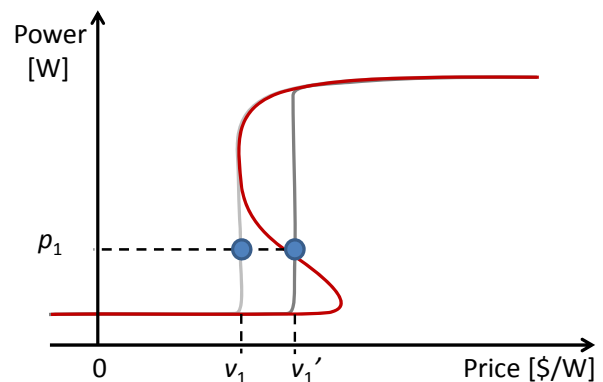


Figure 9: A new hull curve is generated for the non-monotonic cost-function based on the previous market solution (v_1, ρ_1) .

Since all participants in the market use monotonic hull curves, a solution can easily be found. If the solution (v_1, p_1) is now placed in a section where the hull curve does not coincide with the effective cost curve, the correspondent producer has to “reveal” its effective costs (v_1', p_1) in the second round of negotiation.

To this end, a new hull curve is generated. Again it must be monotonic increasing. But the solution of the first round now splits the hull curve in two parts:

- For $v < v_1'$, the curve must again be greater or equal than the part of the effective cost curve that is lower than p_1 and within these constraints as low-valued as possible.
- For $v \geq v_1'$ the curve must be greater or equal to than the effective cost curve or equal to p_1 , again, as low-valued as possible.

Figure 9 illustrates such a new hull curve for a given market equilibrium. The procedure can be iterated for several rounds of negotiation. In general, this iteration scheme cannot be proven to approximate the optimal solution, but since each hull curve will be smaller valued than its predecessor the process is at least bound to converge.

In practice, however, this iteration scheme has at least shown to work very well. Therefore let us illustrate it by an example.

4.6 Example2: Non-monotonic behavior.

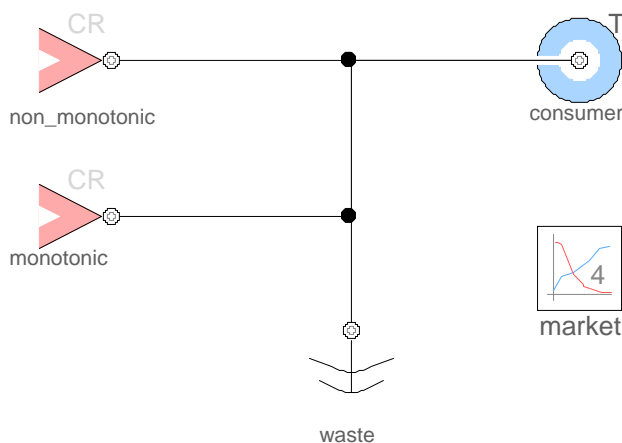


Figure 10: Two sources compete for one consumer. The consumer demand is rising at a constant rate.

In this example, two generators compete to fulfill the power demand of one source. One small generator that is rather inefficient and limited to a small capacity and a large generator that is very efficient for high-load values and very inefficient for low load values. The small generator shall thus be used to overcome the efficiency gap of the large one.

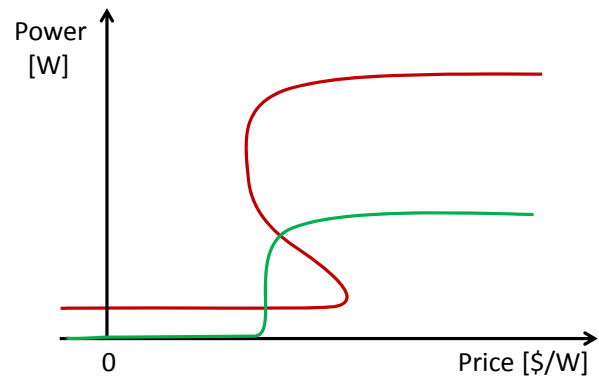


Figure 11: Sketch of the two cost functions for the large (red) and small (green) generator.

Figure 11 sketches the two cost functions and Figure 10 displays the corresponding model diagram. To enable several iterations for the final solution, the price per watt and the power have been implemented as vectors (see Section 3.1). By the parameter n , the number of iterations can be determined. In this case, we choose $n=4$. This means that the model contains now 4 parallel market models that each represents one round of negotiation.

During simulation the power demand is increasing with a constant rate. Figure 12 and Figure 13 presents the results of the simulation for the different rounds of negotiation. We can see the produced power of each generator.

Clearly, in the first round (blue), the large generator pushes aside its smaller counterpart. But in the following rounds of negotiation, the small generator can make its point. The resulting final behavior (magenta) almost leads to a discrete switch as soon as the large generator becomes more efficient as its smaller counterpart. The simulated results reflects an almost optimal behavior.

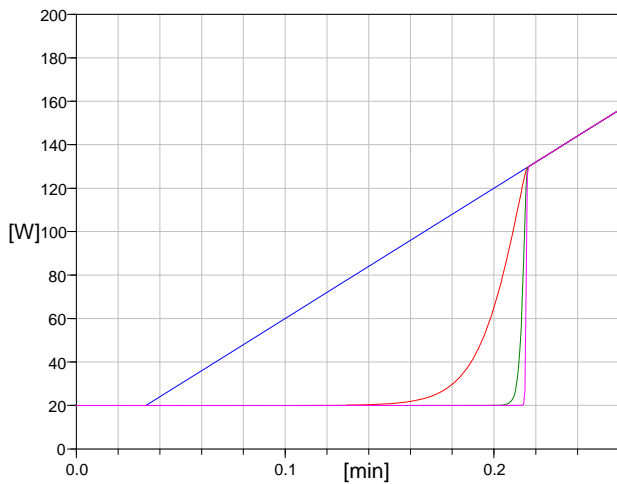


Figure 12: Power output of the large generator for different rounds of negotiation (round 1: blue, round 2: red, round 3: green, round 4: magenta).

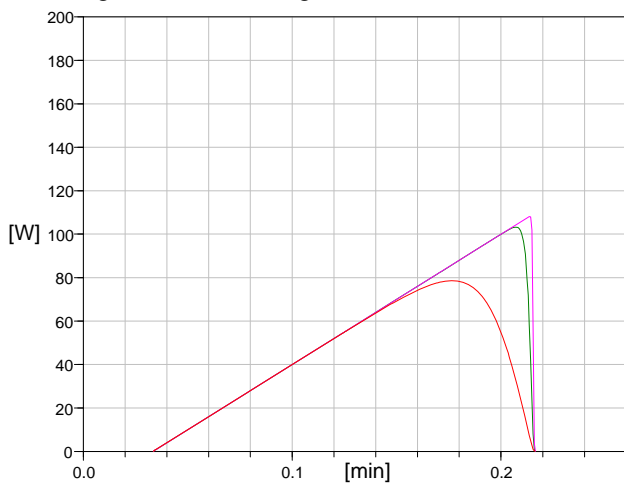


Figure 13: Power output of the small generator for different rounds of negotiation (round 1: zero valued, round 2: red, round 3: green, round 4: magenta).

5 Application Domain: Load Management

5.1 Motivation

A typical load management (e.g. as applied in the electrical system of an aircraft) can cut and reconnect loads depending on its priority. The priorities can directly be translated into prices. Thus low priority loads just pay low prices for a certain amount of power whereas high priority loads pay high prices.

The goal is to get a stable, object-oriented load management function. Thus it is possible to get an implementation very quick and enable an early integration of the function into design process of system to be controlled. Furthermore, modular functionality like dealing with switchable and continuous loads in one system can easily be added.

5.2 Derivation of cost functions

Other than source management, the model of a typical source for load management looks rather different. The focus is on maximum availability of loads and stability, not on energy efficiency. A source function as illustrated in Figure 14 is implemented having linear segments in three areas.

In area I, all loads are on. So there is no special requirement on the function rather than being monotone and continuous. Area III defines the maximum power capacity of the generator by means of a constant value. In this area all controllable loads shall be off. Within area II, cutting of switchable loads and decreasing of continuous loads take part.

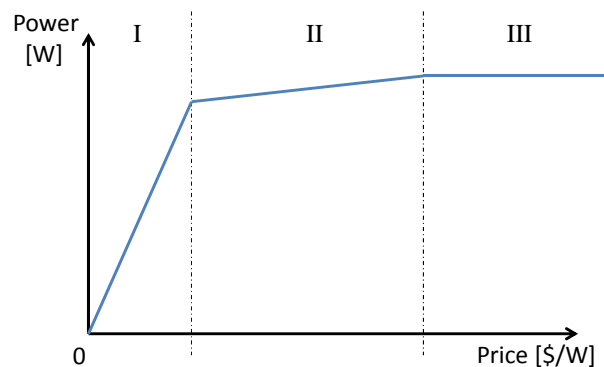


Figure 14: Cost-function of a source having three areas: I – all loads on, II – shedding, decreasing loads with respect to its priority, III – all controllable loads off.

As shown in Figure 15 the cost-functions of switchable and continuous loads are quite equal in principle.

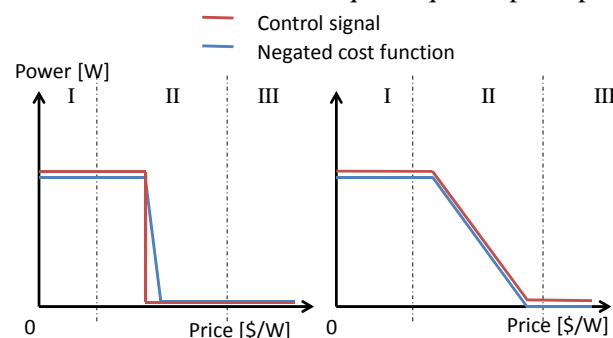


Figure 15: Negated cost functions and control signals of switchable loads (left) and continuous loads (right).

They consist of a full-power area, a linear decreasing area, and a zero-power area. The main difference is the slope of the function. The following inequation applies:

$$\text{slope}(\text{switchable loads}) \gg \text{slope}(\text{continuous loads})$$

Furthermore, the control signal is different for the two types of loads. All switchable loads receive an off-signal, if the current price is not within full-power area whereas all continuous loads receive a continuous power signal as determined in the cost-function. Since the location of the linear decreasing segment is determined via the priority of the loads and a global market model prescribes the location of the areas I, II, and III it can be guaranteed that this linear segment lies entirely in area II.

As the switchable loads are cut at the linear decreasing segment, one must avoid having two loads with the same priority. Otherwise both loads will be cut, even if not needed. Thus, each load should have its own priority.

If there are switchable and continuous loads in one system, multiple rounds of negotiations can be used to determine the power inflow for the continuous loads. This is done via setting a price in a first negotiation round using all cost-functions as described previously for calculating the control signals for the switchable loads. A second and final negotiation round for the continuous loads can then use these discrete control signals and assume all cost-functions of the switchable loads to be constant in all three areas (on or off). Thus less generator-capacity is wasted.

In typical load management systems, there are usually additional restrictions rather than the available generator capacity (e.g. a feeder that limits transmitted power or current to a set of loads). This can be modeled easily by means of a limiter as shown in Table 1. On the output plug, a price can be increased if a prescribed limit is exceeded. The preferred implementation includes qualitatively the same cost-function as for the generator (see Figure 14). At the output plug, a maximum function is applied that defines either the price at the input plug (i.e. from the price coming from the generator) or the price of the limiter. This ensures compliance with the restriction as well as an optimal availability of high priority loads.

5.3 Example

Figure 16 shows a simple setup of a load management model consisting of one source, three feeders (limiters) and six different loads. The model is set up in the same way like the corresponding physical electrical system

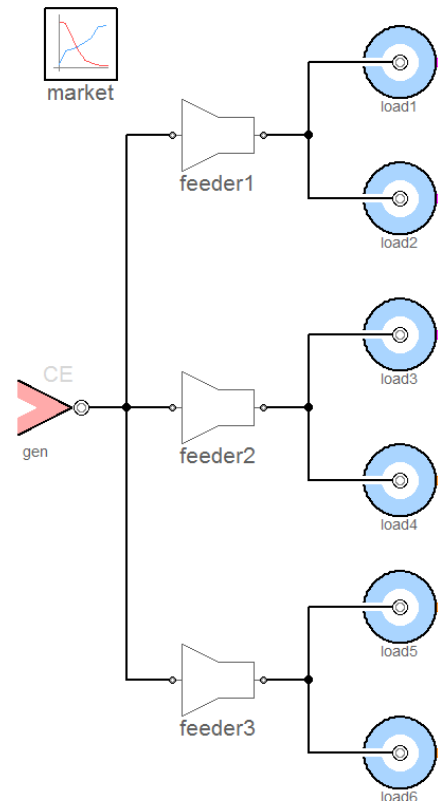


Figure 16: Example of a load management model having one source, 3 feeders and 6 loads (mixed continuous and switchable).

After specifying the nominal values for the source (generator) and the feeders as well as setting the priority of the loads, the load management function is ready to be used. Depending on actual power demand (input not illustrated in the figure), loads will be shed, reconnected, or reduced to comply with all restrictions of the source and limiters.

6 Conclusion and future work

This work represents our first approach towards a market-oriented modeling of energy-management tasks using a Modelica library. The current results look promising and demonstrate the principal functionality of the library. It can be used both for source and load management and also more difficult tasks such as non-monotonic cost functions can be reasonably well handled.

Although, we have analyzed only rather small systems so far, the simulation performance was always very good. We expect thus that the approach is also for feasible for larger systems with hundreds of generators and consumers.

One mayor advantage of having an energy management function directly implemented in Modelica is the easy coupling to the physical system it shall control. This enables an improved development pro-

cess of the system in conjunction with its control function and thus early optimization of both.

In case of source management, certain tasks need to be approached in order to create a solution that is more intuitively applicable for engineers. The import of characteristic curves (based on real data) for the efficiency of generators shall be supported by the library. In addition, the library needs to be tested at a larger set of more realistic examples. Further future potential concerns the modeling of dynamic characteristics. Power generators typically cannot increase their output power at any arbitrary rate. Also storage components like batteries have a dynamic pricing of their energy.

In case of load management, further functionality like variable cost functions shall be added to the library by allowing variable priorities. This enables a more flexible energy management, since the importance and availability of a load can change during operation. In addition, sources like generators can often be overloaded due to their heat capacity. Thus they shall also influence the cost function dynamically. Furthermore, additional elements like switches can be added to allow adaption of the management function in case of a network re-configuration.

One further major step is to combine both sub-libraries in a suitable way. This means to manage priorities of the loads as well as energy efficiency by one cost function. To this end, a more elaborated determination of price according to load priority, energy efficiency, and further restrictions is needed.

References

- [1] Büchner, Stefan. *Energiemanagement-Strategien für elektrische Energiebordnetze in Kraftfahrzeugen*. Dresden, Germany: PhD thesis, faculty of Transportation and traffic sciences Friedrich List, TU Dresden, 2008.
- [2] Brewer, J. W. Bond Graphs of Microeconomic Systems. *Automatic Control Division*, ASME, New York, 1976
- [3] Brewer, J. W. and P. C. Craig. Bilinear, Dynamic Single-ports and Bond Graphs of Economic Systems. *Journal of The Franklin Institute*, No 185, 1991
- [4] Cellier, F. E., *Continuous System Modeling*. Springer Verlag New York, 1991.
- [5] Cellier, F.E., World3 in Modelica: Creating System Dynamics Models in the Modelica Framework, *Proc. 6th International Modelica Conference*, Bielefeld, Germany, Vol.2, pp. 393-400, 2008.
- [6] Engstle, Armin. *Energiemanagement in Hybridfahrzeugen*. Munich, Germany: PhD thesis, Fakultät Elektrotechnik und Informatik, TU München, 2008.
- [7] Odum, H.T, *Ecological and General Systems: An Introduction to Systems Ecology*, Colorado University Press, Colorado, USA, 1994.
- [8] Schlabe, D., and J. Lienig, "Energy Management of Aircraft Electrical Systems - State of the Art and Further Directions," *International Conference on Electrical Systems for Aircraft, Railway and Ship Propulsion (ESARS)*, Italy, October 2012.
- [9] Ygge, Fredrik, *Market-Oriented Programming and its Application to Power Load Management*. Lund, Sweden: PhD thesis, Department of Computer Science, Lund University, 1998.

Modeling and Simulation of a Linear Piezoelectric Stepper Motor in MapleSim

Orysia Soroka

Derek Wright

Orang Vahid

Maplesoft

Waterloo, Ontario, Canada

oosorok@uwaterloo.ca, dwright@maplesoft.com, ovahid@maplesoft.com

Abstract

Devices based on piezoelectric materials have traditionally been modeled in PDE simulation software. These simulations are expensive to create and run. In this paper it is shown that lumped-parameter models of such devices can provide good fidelity with low computational cost. Modelica models of supporting components, along with a system-level model of a linear piezoelectric stepper motor are presented. The simulation results show good agreement with published experimental results. Future research is proposed based on the components and model.

Keywords: Piezoelectric, Linear Motor, MapleSim

1 Introduction

Piezoelectric materials experience mechanical stress under the influence of an electric field and, inversely, produce an electric field with the application of a mechanical stress. Materials that exhibit the piezoelectric effect are used in diverse applications, including a variety of sensors and actuators, and specifically in stepper motors. Detailed PDE simulations of these materials are achievable using simulation software such as COMSOL, but lumped-parameter models suitable for component- and system-level simulations are rare. Developing piezoelectric materials models in Modelica makes modeling and simulation at the system-level possible. A resulting library of parametrically-defined component models, like motors and actuators, would increase the efficiency of modeling and simulating piezoelectric devices routinely deployed in new engineering designs.

In this research, Modelica components implementing piezoelectric material properties, electrostatic forces, and time-varying frictions were developed and integrated into a device-level model of a linear piezoelectric stepper motor. The model is parametric and extensible: the parameters can be changed to suit

application-specific requirements, and nonlinear effects can be easily included.

MapleSim is a Modelica-based system-level modeling and simulation platform provided by Maplesoft [1]. MapleSim simulation results matched those in [2] when similar values were implemented. Most importantly, the relative execution speed of the model permits multi-parameter optimizations not possible in full PDE simulations. This is demonstrated via the investigation of the effects of the motor clamp voltage on velocity using a compiled MapleSim procedure in Maple. Future work is then described.

1.1 Related Work

To the authors' knowledge, there is no formal Modelica library available for piezoelectric materials. However, there have been a variety of disparate works that have implemented piezoelectric models in a lumped-parameter framework. For example, a MEMS library and airbag deployment example including piezoelectric elements were implemented in VHDL-AMS in [3] and [4], respectively. Lumped-parameter models of piezoelectric devices derived from high-order FEM models, are presented in [5], but are not Modelica-specific implementations. They would retain some of the discretized nature of the original FEM models and would therefore be further away from the benefits of using Modelica. In [6], bond graph and equivalent circuit methods are used to model piezoelectric motors. Finally, several tool-independent lumped-parameter physics-based models are presented in Chapter 6 of [7].

2 Linear Motor Operation

Figure 1 shows the configuration and operation of the linear motor. The operation is similar to other slip-stick motors, but is unique in that an electrostatic clamp is used to aid the "stick" portion of the cy-

cle. Periodic waveforms are applied to extend and relax the piezoelectric material along its longitudinal direction, pushing the lead weight along with it. The electrostatic clamp is active during the extension part of the cycle to prevent the motor assembly from slipping along the surface. An abrupt voltage is applied to the piezoelectric material when it is in its extended state and the clamp is deactivated to cause the assembly to retract towards its new center of mass, moving it forward.

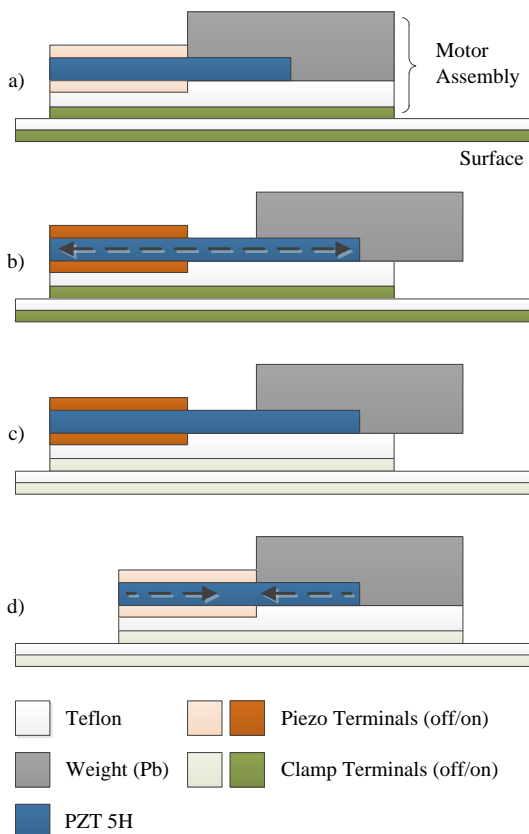


Figure 1: Linear motor configuration and operation. a) The electrostatic clamp is activated. b) The piezoelectric material extends longitudinally with an applied voltage, moving the center of mass to the right. c) The clamp is deactivated. d) The piezoelectric voltage is quickly removed to cause a snapping motion, breaking the static friction between the motor assembly and the surface. The assembly retracts towards its new center of gravity, moving forward.

To model this in MapleSim via Modelica, several new components were needed: A 1D model of the piezoelectric material which couples the electrical and translational domains, an electrostatic clamp that also couples the electrical and translational domains, and a time-varying friction model. Their development is described next.

3 Component Models

In the following sections, variables indicated in bold face correspond to port variables. Numbers in brackets preceding an equation (like (1), for example) indicate equations that appear in the final Modelica component.

These components were first created as MapleSim Custom Components, which directly implement their governing equations developed in Maple. Essentially, the equations are written unsimplified and MapleSim automatically rearranges and manipulates them as needed. Upon creation of the component, Modelica code is auto-generated which was then manually further modified.

3.1 Piezoelectric Material Model

The development of a 1D piezoelectric model relied heavily on Chapter 6 of [7]. The full tensor solution was reduced to the (3,3) direction to select the longitudinal translational mode of operation. Losses and nonlinearities, such as hysteresis, were neglected as a first-order approximation. Such effects can be easily included in the core equations, or included externally using Modelica Standard Library components. Maxwell's equations were accordingly simplified. Refer to Figure 2 for referencing of the port variables.

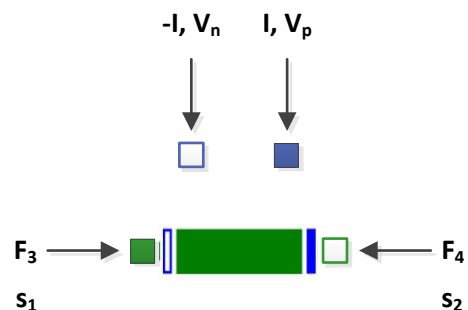


Figure 2: Through- and across-variable references for the piezoelectric component.

In one dimension, the traction (stress) of a piezoelectric material is

$$T = c^D \cdot S - h \cdot D$$

where T is the traction, c^D is the mechanical stiffness of the material, S is the mechanical strain, h is a piezoelectric coupling coefficient with units of V/m, and D is the electrical displacement field. Neglecting inertia, the forces at either end of a slab of length l and area A of this material are

$$F_1 = -A \cdot T|_{z=0}, F_2 = -A \cdot T|_{z=l}$$

noting that traction is referenced positive in the tensile direction. Therefore

$$F_1 = -A(c^D \cdot S - h \cdot D)|_{z=0}$$

The strain can be approximated by taking the first derivative of the material's displacement in Eulerian coordinates, ξ , so that

$$S \cong \frac{\partial \xi}{\partial z}$$

and therefore

$$F_1 = A \left(h \cdot D - c^D \cdot \frac{\partial \xi}{\partial z} \Big|_{z=0} \right)$$

The D field can be replaced with the charge, Q , as follows:

$$\frac{\partial D}{\partial t} = J_{disp}$$

$$\oint J_{disp} \cdot dA = I$$

$$\therefore J_{disp} = \frac{I}{A}$$

$$\therefore \frac{\partial D}{\partial t} = \frac{I}{A}$$

Since

$$(1) I = \frac{dQ}{dt}$$

$$\therefore D = \int \frac{I}{A} \cdot dt = \frac{Q}{A}$$

$$F_1 = h \cdot Q - A \cdot c^D \cdot \frac{\partial \xi}{\partial z} \Big|_{z=0}$$

where J_{disp} is the displacement current and I is the electrical current. Noting that

$$\frac{\partial \xi}{\partial z} \Big|_{z=0} \cong \frac{\xi_2 - \xi_1}{l} = ds$$

$$(2) ds = s_2 - s_1 - l$$

Therefore,

$$(3) F_1 \cong h \cdot Q - A \cdot c^D \cdot \frac{ds}{l}$$

$$(4) F_2 = -F_1$$

To incorporate inertia, one-half of the calculated mass is placed on either side of the piezoelectric material. It is calculated from its density, ρ , length and area. Damping could also be included in these equations, but was not necessary for this particular analysis.

$$(5) F_3 - F_1 = \frac{1}{2} \rho \cdot l \cdot A \cdot \frac{\partial^2 s_1}{\partial t^2}$$

$$(6) F_4 - F_2 = \frac{1}{2} \rho \cdot l \cdot A \cdot \frac{\partial^2 s_2}{\partial t^2}$$

Finally, the terminal voltage, V , can be calculated as the integral of the electrical field, \mathcal{E} , as

$$V = \int_0^l \mathcal{E} \cdot dz$$

Since \mathcal{E} can be defined as a function of D and S via

$$D = e \cdot S + \varepsilon^S \cdot \mathcal{E}$$

where e is the (3,3) element of the piezoelectric stress matrix. It can be defined as

$$e = \varepsilon^S \cdot h$$

where ε^S is the electrical permittivity of the piezoelectric material under constant strain conditions. Therefore,

$$\mathcal{E} = \frac{D}{\varepsilon^S} - h \cdot S \cong \frac{Q}{A \cdot \varepsilon^S} - h \cdot \frac{\partial \xi}{\partial z}$$

and

$$V = \int_0^l \left(\frac{Q}{A \cdot \varepsilon^S} - h \cdot \frac{\partial \xi}{\partial z} \right) \cdot dz$$

$$(7) V = \frac{Q \cdot l}{A \cdot \varepsilon^S} - h \cdot ds$$

where

$$(8) V = V_p - V_n.$$

3.2 Electrostatic Force Model

An electrostatic force model was implemented that couples the electrical and translational domains. Unlike in the piezoelectric model which did incorporate a linear stress-strain relationship, the stress-strain relationship of the dielectric material under the influence of the applied electrostatic force was not included. It is present in the system-level model as a translational spring. This decision was made so that the component could be easily modified as needed. For example, more accurate models would use a translational spring-damper to incorporate losses, and keeping it outside the electrostatic force component facilitates this change. Refer to Figure 3 for referencing of the port variables.

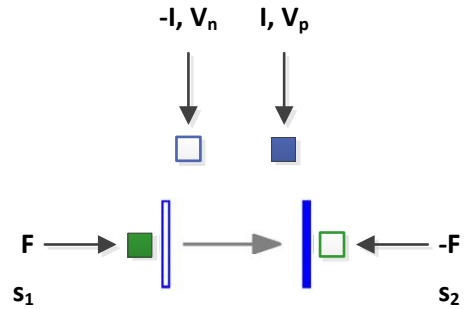


Figure 3: Through- and across-variable references for the electrostatic force component.

Neglecting edge effects, the force between two plates of a parallel capacitor and current are

$$(1) F = \frac{Q \cdot V}{d} = \frac{C \cdot V^2}{d}$$

$$(2) C = \frac{\varepsilon_0 \cdot \varepsilon_r \cdot A}{d}$$

$$(3) I = \frac{d}{dt} (C \cdot V)$$

where

$$(4) d = s_2 - s_1$$

and

$$(5) V = V_p - V_n.$$

3.3 Smooth Time-Varying Friction Model

The purpose of this model was twofold: First, a time-varying friction was needed where the normal force and coefficients were time-dependent. This was due to the electrostatic clamp changing the applied normal force. Second, whereas the standard friction model is discontinuous when transitioning from static to dynamic, a continuous model would produce similar results and would speed simulation time by avoiding events. It also eliminated the need to provide scaling information to the solver to detect events within such a narrow band of operation. Refer to Figure 4 for referencing of the port variables.

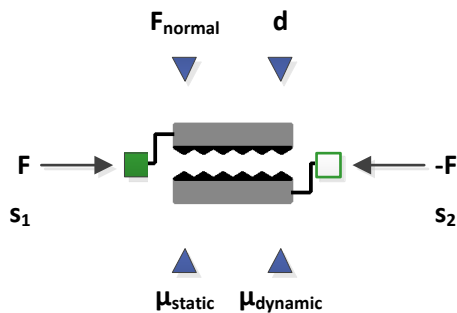


Figure 4: Through- and across-variable references and input signals for the time-varying friction component.

Beginning with the smooth friction model, a sum of two hyperbolic tangents was used to create the approximation.

$$(1) \quad g(x, A_1, A_2) = A_1 \cdot \tanh(c_1 \cdot x) + (A_2 - A_1) \cdot \tanh(c_2 \cdot x)$$

In its intended usage, x would be the relative velocity, A_1 would be the static friction, and $(A_2 - A_1)$ would be the dynamic friction. The coefficients c_1 and c_2 are chosen so that $c_1 > c_2$, which gives the desired function shape. An example is shown in Figure 5 and its similarity to the basic discontinuous friction model should be noted.

Similarly, a smooth step-like function was used to ensure that when non-positive normal force is applied, there is no resultant “negative” friction. Such a function was implemented using

$$(2) \quad h(x) = \frac{1}{2} (\tanh(c_3 \cdot x) + 1)$$

and an example plot is shown in Figure 6.

Using these smooth equations, the friction model is then implemented as

$$(3) \quad s_{rel} = s_2 - s_1$$

$$(4) \quad v_{rel} = \frac{ds_{rel}}{dt}$$

$$(5) \quad F_{static} = \mu_{static} \cdot F_{normal} \cdot h(F_{normal})$$

$$(6) \quad F_{dynamic} = \mu_{dynamic} \cdot F_{normal}$$

$$(7) \quad F = g(v_{rel}, F_{static}, F_{dynamic} - F_{static}) + d \cdot (v_{rel})$$

where d is the damping coefficient.

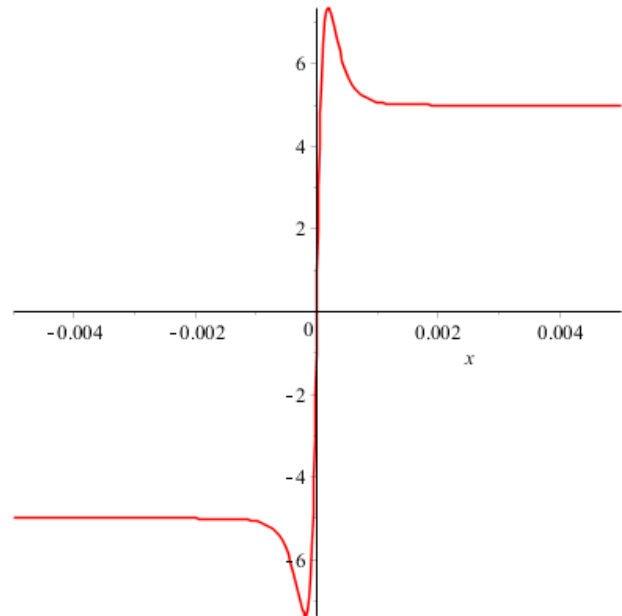


Figure 5: Example plot of the smooth friction model for parameters: $A_1 = 10$, $A_2 = 5$, $c_1 = 10000$, $c_2 = 2500$.

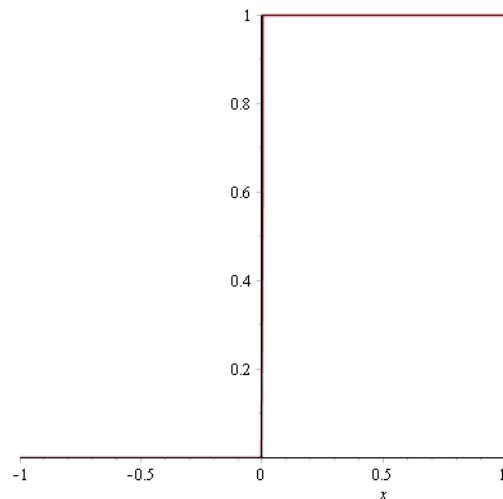


Figure 6: Example plot of the smooth step function for the parameter $c_3 = 10000$.

4 Slip-Stick Motor Model

The three new Modelica components were assembled in MapleSim along with library 1D translational and signal components to create the overall model, shown in Figure 7.

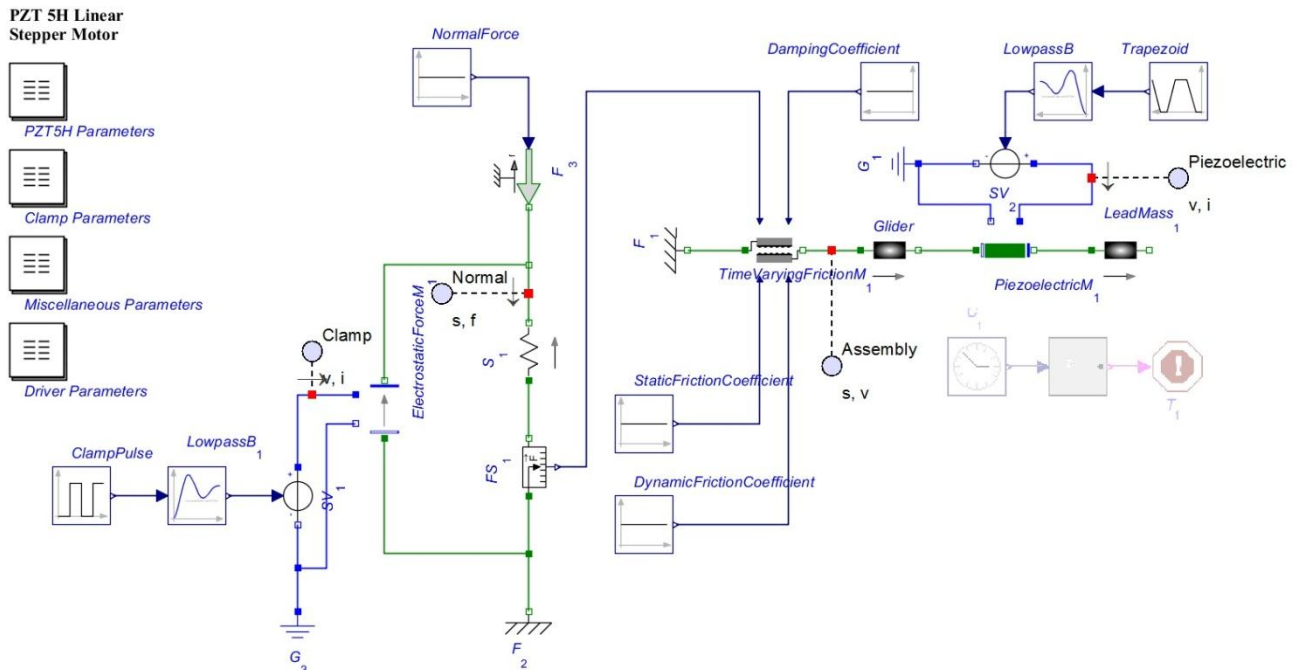


Figure 7: The MapleSim schematic of the parametrically-defined linear motor model.

The model was defined parametrically, using the parameters summarized in [2] as nominal values. Amazingly, the results matched quite well just by using the physical parameters and using some basic assumptions on the undocumented parameters, in particular, the characteristics of the driver waveforms. For example, it is stated in [2] that step sizes of $0.07 - 1.1 \mu\text{m}$ were observed for piezoelectric voltages of $60 - 340 \text{ V}$. The MapleSim model achieved $0.061 - 0.371 \mu\text{m}$ step sizes for the same applied voltages without any tuning or optimization of the unknown parameters. Adjusting the magnitude of the clamp voltage and frequency cutoff of the filters are two of the easiest ways of changing the step size to help it match the experimental results. Therefore, the MapleSim model represents a reasonable approximation to the system behavior without the burden of a full PDE solution.

4.1 MapleSim Model and Preliminary Results

As stated previously, the model matches the experimental results quite well and provides additional degrees of freedom to back-fit to the available data. Figure 8 and Figure 9 show the applied driver signals and resulting motor motion, respectively. A comparison to the results in [2] shows good qualitative and numerical agreement.

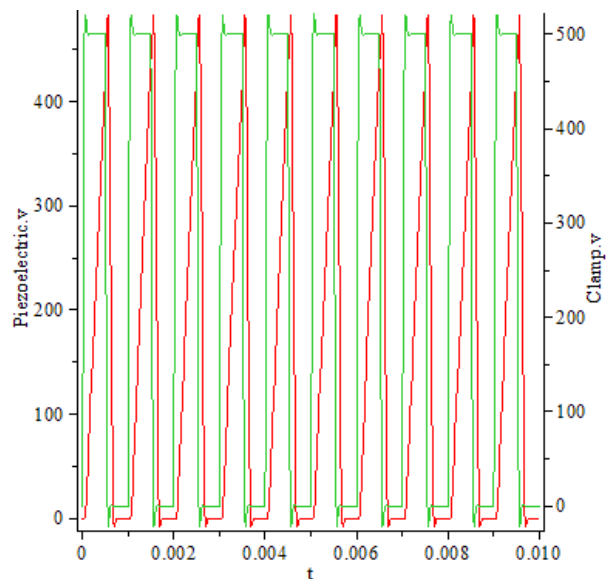


Figure 8: 1 kHz clamp (green) and piezoelectric (red) drive voltage signals. The slight overshoot is due to a low-pass filter set to 10 kHz to limit discontinuities present in the simulation.

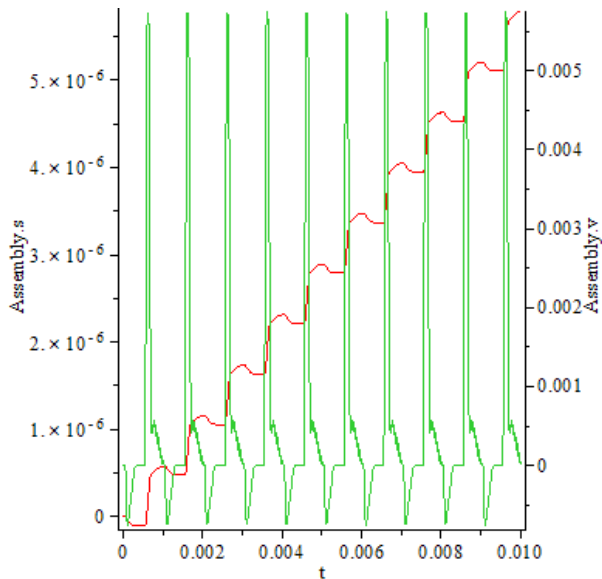


Figure 9: Plots of the position (red) and velocity (green) versus time of the linear motor.

4.2 Platform for Optimization

One of the goals of this research is to demonstrate the value of system-level models of devices that traditionally have only been modeled in PDE software. As an example of the execution speed and optimizations possible, consider the results in Figure 10, and further summarized in Figure 11. They show the position versus time and velocity versus V_{clamp} results for 100 simulations, respectively. When comparing to the results presented in Fig. 10 in [2], it can be seen that the results are quite consistent.

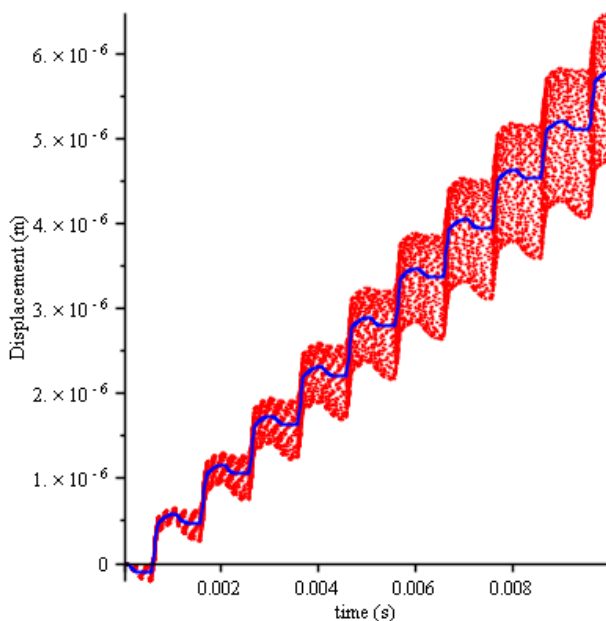


Figure 10: Position versus time plots for V_{clamp} values from 0 to 1000 V. The nominal value, $V_{clamp} = 500$ V, is shown in blue.

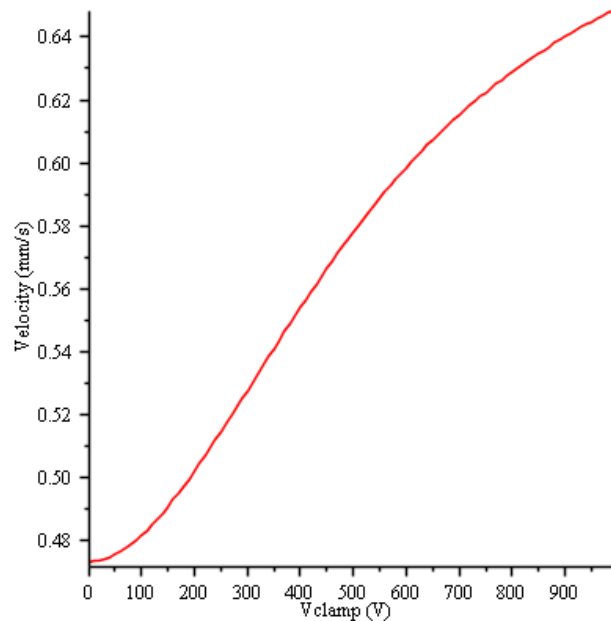


Figure 11: Calculated average velocity values for various V_{clamp} . Note how the electrostatic clamp improves the speed of the motor by preventing reverse motion during extension of the piezoelectric material.

The per-simulation execution time was 63.8 ms on a modest Intel Core2 Duo CPU running at 2.80 GHz. Similar results would take a tremendous amount of time in PDE simulation software. Though the PDE results would arguably be more accurate, the marginal accuracy is of questionable value in light of the orders of magnitude increase in simulation time.

5 Conclusions and Further Research

This paper has demonstrated the creation of a linear piezoelectric stepper motor in MapleSim. To produce the motor model, three new components were created and their derivations were documented. Initial results correlate well with published experimental results, indicating that lumped-parameter system-level models may provide a new platform for development and optimization of such devices.

The follow-up research currently underway involves multi-parameter optimizations in a multi-threaded, multicore architecture in Maple. The goal would be to demonstrate that fast MapleSim models can be used to optimize for goals like motor speed and power consumption, as well as to more accurately fit the experimental data. This will be accomplished directly in Maple via its threads and grid computing capabilities, and in Optimus, a global op-

timization and design-of-experiments package by Noesis [8].

Using the piezoelectric material model as a starting point, further developments include a full multi-body (6 DoF) model of the material behavior. It is created using the full tensor description of the piezoelectric material. This will enable the development of novel devices using torsional modes, and a more accurate look into the behavior of existing devices, like the motor presented in this paper.

References

- [1] www.maplesim.com
- [2] Judy J W, Polla D L, and Robbins W P. A Linear Piezoelectric Stepper Motor With Submicrometer Step Size and Centimeter Travel Range. *IEEE Trans. UFFC*, Vol. 37, No. 5, 1990.
- [3] Schwarz P, and Schneider P. Model Library and Tool Support for MEMS Simulation. *SPIE Proc. Microelectronic and MEMS Technology*, Vol. 4407, 2001.
- [4] Pecheux F, Allard B, Lallement C, Vachoux A, and Morel H. Modeling and Simulation of Multi-Discipline Systems Using Bond Graphs and VHDL-AMS. *Proc. ICBGM*, 2005.
- [5] Gentili L, Bassi L, Macchelli A, Melchiorri C, and Borsari R. Model Reduction for High-Order Port-Hamiltonian Systems. Application to Piezo-Electric Systems. *Proc. IEEE Conf. Decision and Control*, 2009.
- [6] Essalam B A, and Mabrouk K. Generation of analytical redundancy relations for fault detection and isolation of ultrasonic linear motor. *Nature & Technology*, Vol. 4, 2011.
- [7] Cobbold R S C. *Foundations of Biomedical Ultrasound* (New York: Oxford University Press). 2007.
- [8] www.noessolutions.com

Magnetic Hysteresis Models for Modelica

Johannes Ziske, Thomas Bödrich

Technische Universität Dresden, Institute of Electromechanical and Electronic Design

01062 Dresden, Germany

Johannes.Ziske@tu-dresden.de

Thomas.Boedrich@tu-dresden.de

Abstract

Modelica models for transient simulation of magnetic hysteresis are currently being developed at Technische Universität Dresden. This paper gives an overview about the present state of the work. Two hysteresis models have been implemented so far in Modelica and are currently optimised and tested: the rather simple but efficient Tellinen model and the more complex and accurate Preisach model. Utilisation of the Tellinen model together with components of the Modelica.Magnetic.FluxTubes library is exemplarily shown with transient simulation of a three-phase autotransformer. Additionally, an efficient implementation of the Preisach model is described and a first comparison between the Tellinen and the classical Preisach hysteresis model is presented. It is planned to include the developed hysteresis models into the above-mentioned FluxTubes library after their further optimisation and validation with own measurements. These models will especially allow for the estimation of iron losses and for accurate computation of saturation behaviour during Modelica-based design of electromagnetic components and systems. This becomes increasingly important with the growing requirements regarding energy efficiency and mass power densities of such systems.

Keywords: magnetic hysteresis, lumped magnetic network; hysteresis model; Tellinen; Preisach; iron losses; Modelica.Magnetic.FluxTubes library

1 Introduction

The Modelica.Magnetic.FluxTubes library included in the Modelica Standard Library [1] is intended for rough design and system simulation of magnetic components and devices, e.g. actuators, motors, transformers or holding magnets [2, 3]. This library is based on the well-established concept of magnetic flux tubes, which enables modelling of magnetic fields with lumped networks [4].

At present, ferromagnetic hysteresis is not considered in the above-mentioned library. However, the prediction of losses due to static (ferromagnetic) and dynamic (eddy current) hysteresis becomes more and more important during the design of electromagnetic components. This is due to the increasing demands on energy efficiency of electromagnetic systems and due to increasing power densities of those systems. Prominent examples for this engineering trend are electromobility and more electric aircraft, where the necessity of high mass power densities and loss power minimisation are obvious.

In general, the reliable prediction of hysteresis-related losses with lumped magnetic network models is difficult and demanding and has been a topic of research for a long time. Simplified empirical equations for loss calculation, e.g. the well-known Steinmetz formula [5] are based on time-harmonic flux densities of known magnitude and frequency [6]. The delayed penetration of magnetic fields into bulk and laminated ferromagnetic materials can be approximated in lumped magnetic networks with Cauer circuits [7].

Transient simulation of magnetic hysteresis in lumped magnetic network models is possible with dedicated hysteresis models. Well-known such models are for example the phenomenological one published by Preisach in 1935 [8], the physical model developed by Jiles and Atherton [9] or the comparatively simple model developed by Tellinen [10]. Those models are currently analysed at Technische Universität Dresden, and selected hysteresis models are implemented in Modelica for inclusion into the Modelica.Magnetic.FluxTubes library.

The Tellinen hysteresis model and the Preisach model have been implemented and are currently tested and optimised. Theory and Modelica implementation of these two models and their utilisation in components of the Modelica.Magnetic.FluxTubes library will be presented in the following sections. It must be noted that this is a report about work in progress rather than a final presentation of the projected Modelica.Magnetic.FluxTubes library extension. Both

implemented hysteresis models are still subject to optimisation and validation, e.g. with measurements.

2 The Tellinen Hysteresis Model

2.1 Theory

The hysteresis model developed by Tellinen is thoroughly described in [10]. The big advantage of this model is its simplicity. Thus, it is well suited for fast simulations when used in lumped magnetic network models. It works without information about the history of the magnetic field strength H in ferromagnetic components and can completely be configured with the limiting increasing and decreasing branches $\lambda_i(H)$ and $\lambda_d(H)$, respectively, of the limiting hysteresis loop of a ferromagnetic material (Figure 1).

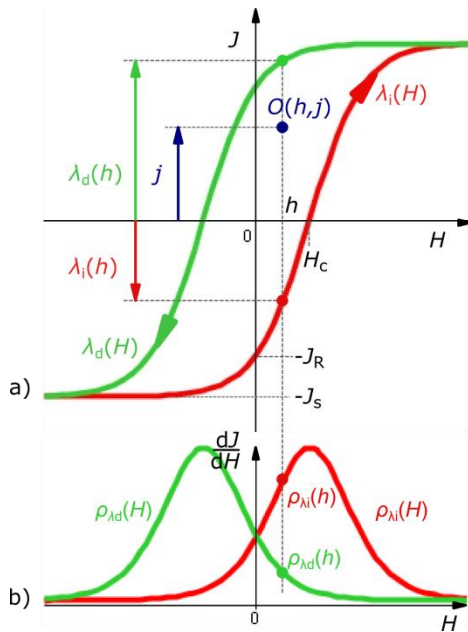


Figure 1: Limiting increasing and decreasing branch $\lambda_i(H)$ and $\lambda_d(H)$, respectively, of a hysteresis loop with magnetic polarization J and magnetic field strength H (a) and corresponding slope functions $\rho_{\lambda_i}(H)$ and $\rho_{\lambda_d}(H)$ (b).

Together with the corresponding slope functions $\rho_{\lambda_i}(H)$ and $\rho_{\lambda_d}(H)$ the actual slope ρ_j at the operating point $O(h, j)$ can be determined as

$$\rho_j = \frac{dj}{dH} = \begin{cases} \frac{\lambda_d(h) - j}{\lambda_d(h) - \lambda_i(h)} \cdot \rho_{\lambda_i}(h) & \text{for } dH > 0 \\ \frac{j - \lambda_i(h)}{\lambda_d(h) - \lambda_i(h)} \cdot \rho_{\lambda_i}(h) & \text{for } dH < 0 \\ 0 & \text{else.} \end{cases} \quad (1)$$

Thus, the time-based slope of j can be easily computed at every integration step to

$$\frac{dj}{dt} = \frac{dj}{dH} \cdot \frac{dH}{dt} = \rho_j \frac{dH}{dt}. \quad (2)$$

Hence the slope of the magnetic flux density db/dt of

$$\frac{db}{dt} = (\rho_j + \mu_0) \cdot \frac{dH}{dt} \quad (3)$$

μ_0 is the slope db/dh of the limiting hysteresis loops within the saturation region.

2.2 Implementation in Modelica

The Tellinen model described above was integrated into a reluctance element of the `Modelica.Magnetic.FluxTubes` library, and can thus similarly be used in electromagnetic network models (in [2] the magnetic library is explained in detail). The reluctance model can be configured with the cross section and the length of a ferromagnetic core and the limiting hysteresis loop of the core material. On the one hand hysteresis loops can be defined by the hyperbolic tangent function and definition of the three parameters J_S (saturation polarization), J_R (remanence) and H_C (coercivity) (see Figure 1a). On the other hand table data can be used to define the increasing and decreasing hysteresis branches. Thus, almost arbitrary hysteresis loops can easily be implemented and also easily be derived from measurements. In addition a small experimental library was built using exemplary table data of some common ferromagnetic materials (Figure 2).

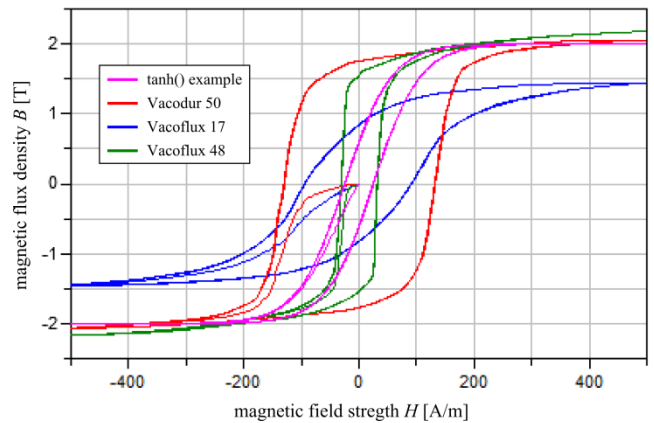


Figure 2: Exemplarily simulated limiting hysteresis loops: curve 1 described by a hyperbolic tangent function and curves 2 to 4 described by tabular $B(H)$ data extracted from [11].

2.3 Autotransformer as an Example

The implemented Tellinen hysteresis models were tested with a simple electromagnetic network model of a three-phase autotransformer. A sketch of the EI-shaped ferromagnetic core of the transformer with indicated corresponding network elements is shown in Figure 3a and the complete electromagnetic network model in Figure 3b.

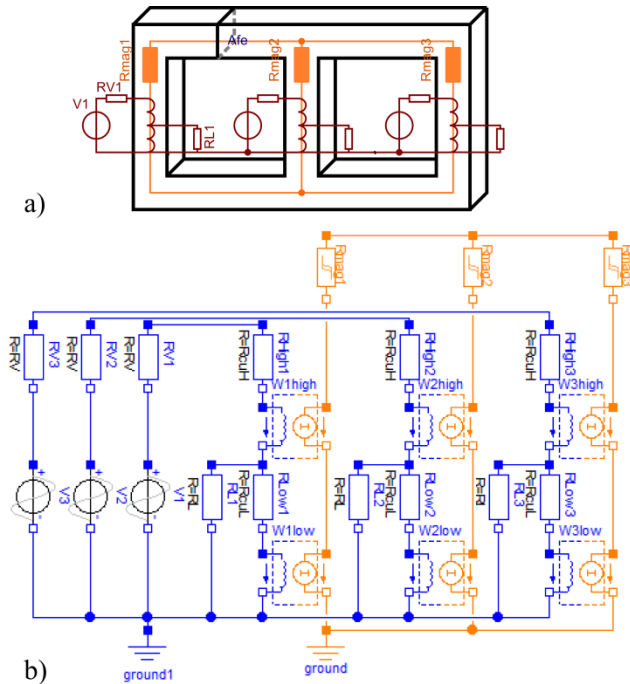


Figure 3: Sketch of a three-phase autotransformer with an EI-shaped ferromagnetic core (a) and corresponding simple electromagnetic network model with hysteresis elements representing the transformer core (b).

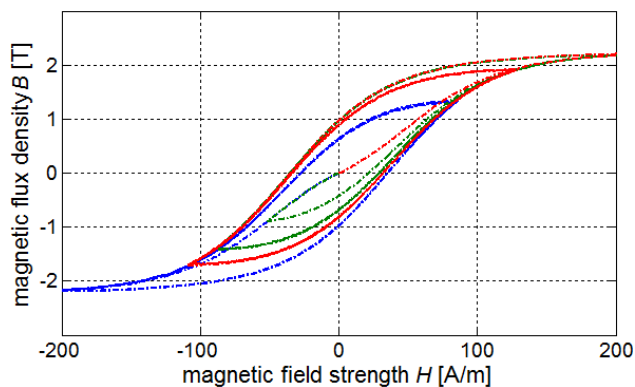


Figure 4: Simulated magnetic flux densities B vs. magnetic field strength H of the three hysteresis elements Rmag1 (blue), Rmag2 (red) and Rmag3 (green) representing the three transformer legs.

Transient oscillations of the magnetic flux densities in the three transformer legs after power-on are exemplarily shown in Figure 4. Selected corresponding voltages and currents are depicted in Figure 5.

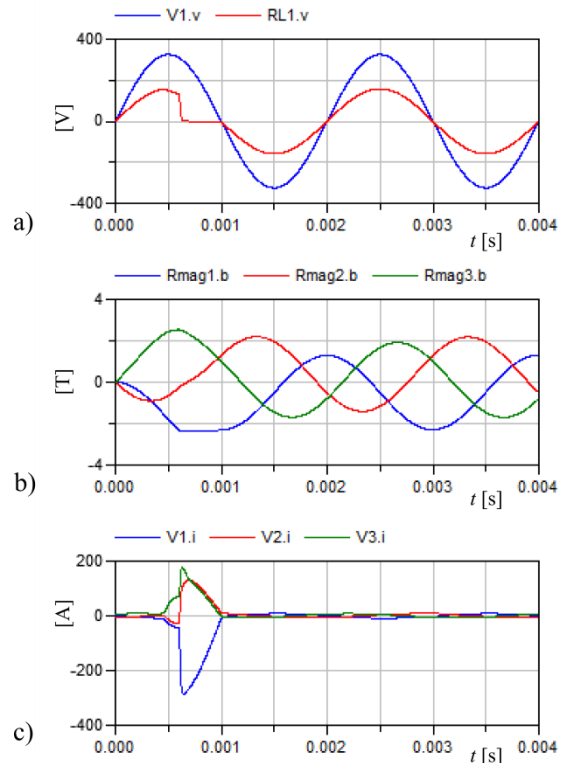


Figure 5: Results of the autotransformer simulation: source voltage $V1.v$ and voltage drop $RL1.v$ of load resistance (a), magnetic flux densities of the three hysteresis elements $Rmag1.b$ to $Rmag3.b$ (b) and source currents $V1.i$ to $V3.i$.

3 The Preisach Hysteresis Model

3.1 Overview on the Classical Preisach Model

In this section a very short overview on the classical Preisach model is given. More detailed information on this model can be found e.g. in [12]. The Preisach model describes the behaviour of an output signal $j(t)$ in dependence on an input signal $h(t)$ and on its history. Here, $j(t)$ and $h(t)$ are the magnetic polarisation of a ferromagnetic material and the magnetic field strength, respectively. The model assumes an infinite set of elementary hysteresis operators $\gamma_{\alpha\beta}$. The operators' output $\gamma_{\alpha\beta}h(t)$ can only hold the polarisation values of -1 or +1 dependent on the upper and lower switching limits α and β , on the input signal $h(t)$ and on its history. The behaviour of $\gamma_{\alpha\beta}h(t)$ is shown in Figure 6. It is defined as

$$\gamma_{\alpha\beta}h(t) = \begin{cases} -1 & \text{for } h(t) \leq \beta \\ +1 & \text{for } h(t) \geq \alpha \\ \text{previous} & \text{else.} \end{cases} \quad (4)$$

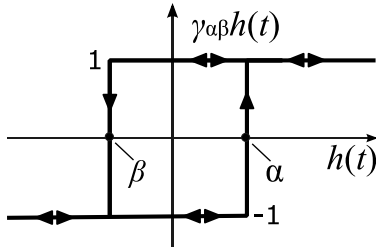


Figure 6: Elementary Preisach operator $\gamma_{\alpha\beta}$ (hysteresis).

The upper switching limit of each operator is always greater than or equal to the lower limit ($\alpha \geq \beta$). Thus, the switching limits α and β span a right triangular region, often referred to as Preisach plane (Figure 7).

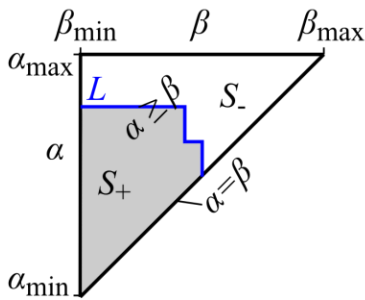


Figure 7: Preisach plane.

For each point (α, β) on this plane exactly one elementary hysteresis operator $\gamma_{\alpha\beta}$ exists with upper and lower switching limits α and β , respectively. The Preisach distribution function $P(\alpha, \beta)$ gives a weight to all operators in the region $\alpha \geq \beta$ and is 0 out of that region. Thus, the output polarisation $j(t)$ of the system results in

$$j(t) = J_S \cdot \iint_{\alpha \geq \beta} P(\alpha, \beta) \cdot \gamma_{\alpha\beta} h(t) d\alpha d\beta \quad (5)$$

(J_S saturation polarisation). An exemplary Preisach distribution function is shown in Figure 8.

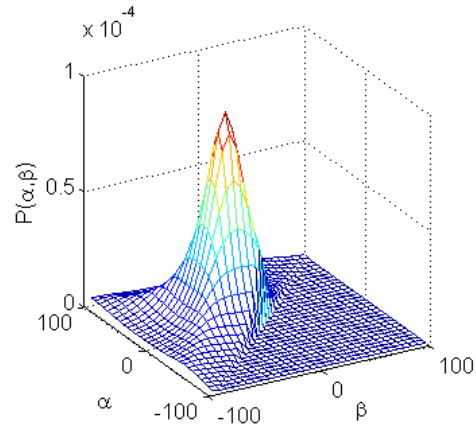


Figure 8: Exemplary Preisach distribution function $P(\alpha, \beta)$ defined over the Preisach plane ($\alpha \geq \beta$).

The Preisach plane can be divided into two regions S_+ and S_- in which all operator outputs $\gamma_{\alpha\beta}h(t)$ are in +1 and -1 state, respectively (Figure 7). Together with Eq. (5) this leads to

$$j(t) = J_S \left(\iint_{S_+(t)} P(\alpha, \beta) d\alpha d\beta - \iint_{S_-(t)} P(\alpha, \beta) d\alpha d\beta \right). \quad (6)$$

With the integral of $P(\alpha, \beta)$ over the region $\alpha \geq \beta$

$$\begin{aligned} \iint_{\alpha \geq \beta} P(\alpha, \beta) d\alpha d\beta &= \iint_{S_+(t)} P(\alpha, \beta) d\alpha d\beta \\ &+ \iint_{S_-(t)} P(\alpha, \beta) d\alpha d\beta = 1 \end{aligned} \quad (7)$$

being equal to 1, Eq. (6) leads to

$$j(t) = J_S \cdot \left(2 \cdot \iint_{S_+(t)} P(\alpha, \beta) d\alpha d\beta - 1 \right). \quad (8)$$

3.2 Implementation in Modelica

In general, the double integral of applied Preisach distribution functions $P(\alpha, \beta)$ cannot be expressed analytically. For that reason the numerical solution of Eq. (8) at every iteration step would be very computationally expensive. Thus, a more efficient calculation method has to be found in order to implement applicable magnetic network components in Modelica.

The evolution of both regions $S_+(t)$ and $S_-(t)$ due to a varying input signal $h(t)$ can easily be visualized in the Preisach plane (Figure 9) [12]. The hypotenuse of the Preisach plane defines the $\alpha = \beta$ line. The input signal $h(t)$ moves as a point along that line if $\alpha_{\min} < h(t) < \alpha_{\max}$.

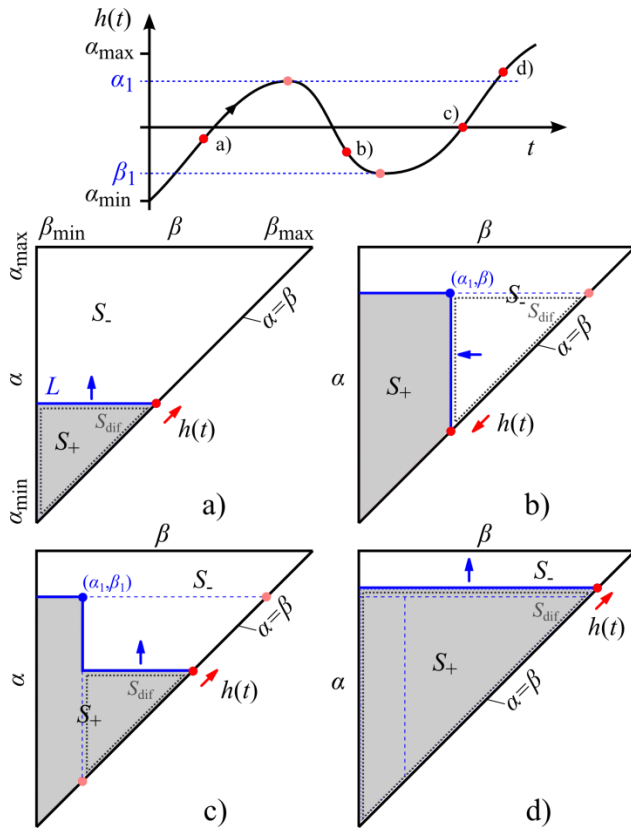


Figure 9: Geometric interpretation of the time-based evolution of the regions $S_+(t)$ and $S_-(t)$ in dependence on the input signal $h(t)$.

Starting from negative saturation (all operators are in -1 state and the whole Preisach plane is filled out by the S_- region) an increasing input moves a horizontal line L (border between S_- and S_+) towards the positive direction of the α -axis, expanding the S_+ region (Figure 9a). When $h(t)$ changes direction the maximum value is stored in α_1 and L is extended by a vertical line moving towards negative direction of the β axis, hereby shrinking again the S_+ region (Figure 9b). If $h(t)$ increases again, the point (α_1, β_1) is fix and β_1 is also stored. Dependent on the course of the input signal a corresponding number n of corner points (α_i, β_i) must be stored. Figure 9c and d show the wiping out of stored points when $h(t)$ becomes larger than the α value of any stored point (α_i, β_i) . Then this point can be deleted since it doesn't contribute any longer to the border between $S_+(t)$ and $S_-(t)$. A similar event occurs when $h(t)$ becomes smaller than the last stored β_i value. Dependent on the number n of stored points, the region S_+ , over which $P(\alpha, \beta)$ must be integrated, becomes more and more complex. However, it can be shown that there is a single triangular region S_{dif} (dotted triangles in Figure 9a to d) for which applies

$$\frac{d}{dt} \iint_{S_+(t)} P(\alpha, \beta) d\alpha d\beta = \frac{d}{dt} \iint_{S_{dif}(t)} P(\alpha, \beta) d\alpha d\beta. \quad (9)$$

Thus, Eq. (8) and (9) lead to

$$\frac{dj(t)}{dt} = 2 \cdot J_S \cdot \frac{d}{dt} \iint_{S_{dif}(t)} P(\alpha, \beta) d\alpha d\beta. \quad (10)$$

S_{dif} belongs to S_+ for increasing $h(t)$ and to S_- for decreasing $h(t)$. It's hypotenuse is part of the $\alpha = \beta$ line of the Preisach plane and thus S_{dif} can be written as difference of the two regions S_1 and S_2 , both having their lower left vertexes at the point $(\alpha_{min}, \beta_{min})$ (Figure 10). This allows to evaluate the integral of $P(\alpha, \beta)$ over the region S_{dif} by two integrals with the same lower integration limits α_{min} and β_{min} respectively:

$$\begin{aligned} \iint_{S_{dif}} P(\alpha, \beta) d\alpha d\beta &= \underbrace{\int_{\beta=\beta_{min}}^{\beta'_2} \int_{\alpha=\alpha_{min}}^{\alpha'_2} P(\alpha, \beta) d\alpha d\beta}_{\iint_{S_1} P(\alpha, \beta) d\alpha d\beta} \\ &\quad - \underbrace{\int_{\beta=\beta_{min}}^{\beta'_1} \int_{\alpha=\alpha_{min}}^{\alpha'_2} P(\alpha, \beta) d\alpha d\beta}_{\iint_{S_2} P(\alpha, \beta) d\alpha d\beta}. \end{aligned} \quad (11)$$

With $\alpha_{min} = \beta_{min} = \text{const.}$, S_{dif} is completely defined by the integration limits $\alpha'_2, \beta'_1, \beta'_2$. Figure 10 shows the integration limits for increasing and decreasing $h(t)$ respectively and their variation due to a change of the input signal $h(t)$.

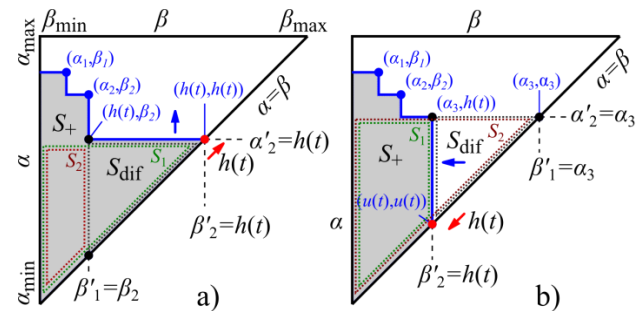


Figure 10: Integration limits α'_2, β'_1 and β'_2 of the region S_{dif} for increasing (a) and decreasing (b) input signal $h(t)$.

From the integral

$$I_P(\alpha', \beta') = \int_{\beta=\beta_{min}}^{\beta'} \int_{\alpha=\alpha_{min}}^{\alpha'} P(\alpha, \beta) d\alpha d\beta \quad (12)$$

and Eq. (11) follows

$$\iint_{S_{diff}} P(\alpha, \beta) d\alpha d\beta = I_P(\alpha'_2, \beta'_2) - I_P(\alpha'_2, \beta'_1). \quad (13)$$

With Eq. (10) and (13) one obtains

$$\frac{dj(t)}{dt} = 2 \cdot J_S \cdot \frac{d}{dt} (I_P(\alpha'_2, \beta'_2) - I_P(\alpha'_2, \beta'_1)). \quad (14)$$

In the Preisach hysteresis model implemented in Modelica, the integral I_P of the Preisach distribution function $P(\alpha, \beta)$ is numerically computed only once at the start of a simulation run for discrete grid points and stored in a two-dimensional array A_{IP} . All values of I_P between the grid points of A_{IP} can then be obtained by bilinear interpolation of adjacent A_{IP} values. This is an enormous reduction of the computational effort, namely from the numerical solution of the double integral of $P(\alpha, \beta)$ to two table look-ups and bilinear interpolations of I_P values in the array A_{IP} (see Eq. (14)). Figure 11 shows the values of A_{IP} for the exemplary Preisach distribution function depicted in Figure 8.

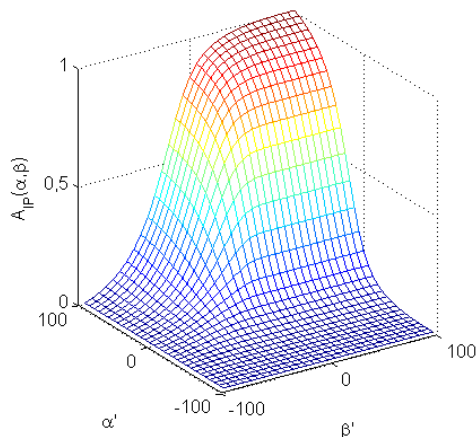


Figure 11: Array data A_{IP} of the integral of the Preisach distribution function $P(\alpha, \beta)$ shown in Figure 8.

3.3 First Simulation Results

A simple network model of an inductor with a closed ferromagnetic core was used for first tests of the implemented Preisach hysteresis model (Figure 12).

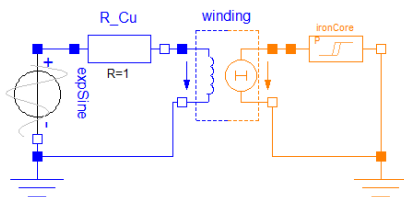


Figure 12: Simple electromagnetic model of an inductor with closed ferromagnetic core for testing of the Preisach hysteresis model.

Simulation results, especially the simulated $B(H)$ hysteresis of the iron core, are shown in Figure 13. The increasing exponential sine voltage causes growing hysteresis loops. The resulting $B(H)$ loops are not centered around the origin, because the flux density B of this simulation starts for $H = 0$ A/m at negative remanence.

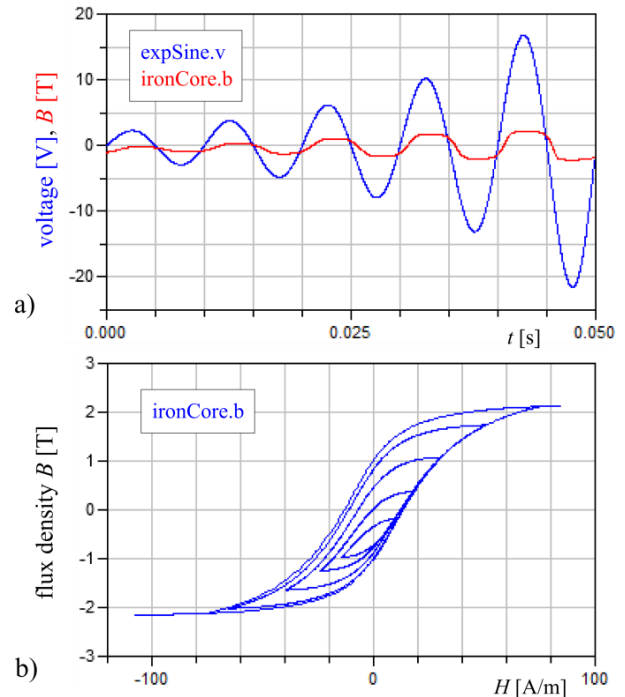


Figure 13: Simulation results of the inductor model: source voltage `expSine.v` and flux density `ironCore.b` in the core (a) and $B(H)$ plot of the growing hysteresis loops in the iron core (b).

4 Model Comparison

To show the different behaviour between the classical Preisach and the Tellinen hysteresis model two simulations were carried out. An identical magnetic field strength $H(t)$ was applied to the input of both hysteresis elements, which were configured to have equal limiting hysteresis loops. The models output characteristics $B(H)$ were then plotted together in one diagram. In the first simulation a decreasing exponential sine wave was used as input signal. The corresponding simulation results are shown in Figure 14. Only small differences in the models output are obvious. The different behaviour can be seen more clearly in the results of the second simulation, in which a slightly more complex input signal of two superposed sine waves of different amplitude and frequency (Figure 15a) was applied. The $B(H)$ characteristics in Figure 15b show the deviation between both models, especially in the region of the minor

loops. In contrast to the Tellinen model, the minor loops of the classical Preisach model are closed.

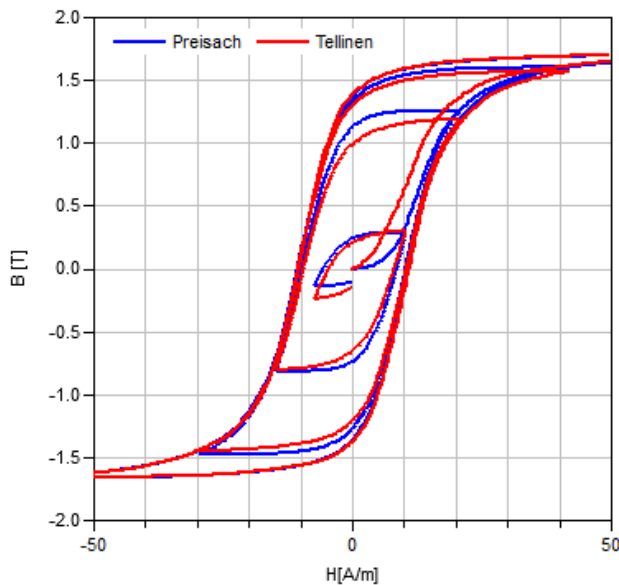


Figure 14: $B(H)$ characteristics of the Preisach and the Tellinen hysteresis model for a decreasing exponential sine wave input signal $H(t)$.

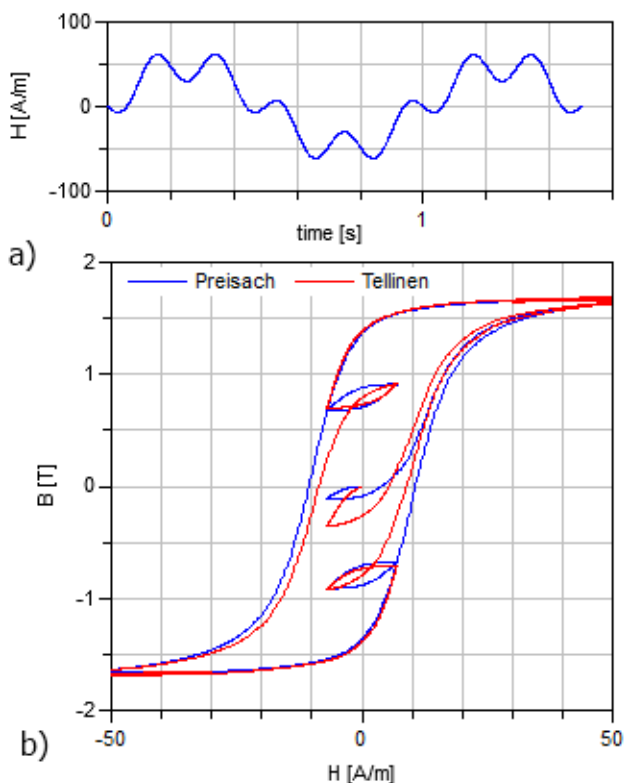


Figure 15: Output of the Preisach and Tellinen model (b) for the identical input signal (a).

Due to the significantly higher computational effort for the Preisach model the network simulation with the Tellinen model performs a lot faster. Dependent on the fineness of the mesh of the discretised Preisach integral, a simulation with one Preisach hysteresis element takes about 3 to 8 times as long as a similar simulation with a Tellinen hysteresis network element.

5 Summary and Outlook

Two different magnetic hysteresis models have been implemented in Modelica: the simple but efficient model developed by Tellinen and the more accurate but complex Preisach model. For latter model, a particular simple and efficient Modelica implementation was derived, hereby reducing the effort for numerical calculation of a double integral over portions of the Preisach plane to two bilinear interpolations in a table.

Utilisation of the Tellinen model together with components of the Modelica.Magnetic.FluxTubes library was exemplarily shown with transient simulation of a three-phase autotransformer.

With further work, the developed hysteresis models will be optimised and tested. Estimation of hysteresis losses from simulated hysteretic behaviour will be implemented. Those simulated iron losses will be provided to a conditional heat port and thus can be input to subsequent thermal simulations, e.g. with models built from Modelica.Thermal.HeatTransfer. Further improvements of the developed hysteresis models will focus on proper initialisation as well as on numerical stability and computational efficiency. If reasonable, the well-known Jiles-Atherton model of magnetic hysteresis will be also implemented. All implemented hysteresis models will be compared with regard to behaviour, accuracy and computation time.

For model validation, measurements of the magnetic properties of selected magnetically soft materials according to EN 60404 are planned. A measurement setup utilising a highly accurate electronic fluxmeter is currently realised. With data obtained from these measurements, the materials sublibrary of Modelica.Magnetic.FluxTubes will be extended and improved. For the Preisach hysteresis model a corresponding parameter identification needs also to be developed for fitting the model behaviour to literature or measured hysteresis data.

6 Acknowledgement

The authors would like to thank the Clean Sky Joint Technology Initiative for funding of the presented work within Project No. 296369 MoMoLib “Modelica Model Library Development for Media, Magnetic Systems and Wavelets”.

References

- [1] Modelica Association, Modelica Standard Library, <https://www.modelica.org/libraries/-Modelica> (May 11, 2012).
- [2] T. Bödrich and T. Roschke, A Magnetic Library for Modelica, in Proc. of the 4th International Modelica Conference, 2005, pp. 559–565.
- [3] T. Bödrich, Electromagnetic Actuator Modelling with the Extended Modelica Magnetic Library, Proc. of 6th Int. Modelica Conf., Bielefeld, Germany, March 3-4, pp. 221–227, 2008.
- [4] H. Roters, *Electromagnetic Devices*. New York: John Wiley & Sons, 1941.
- [5] C. Steinmetz, Hysteresis loss, *Electrician* 26, p. 261 ff., 1891.
- [6] T. Roschke, Entwurf geregelter elektromagnetischer Antriebe für Luftschieber, ser. Fortschritt-Berichte VDI. VDI Verl., 2000.
- [7] D. Ribbenfjård, *Electromagnetic Modelling Including the Electromagnetic Core*, Ph.D. dissertation, KTH Royal Institute of Technology, Stockholm, 2010.
- [8] F. Preisach, Über die magnetische Nachwirkung, *Zeitschrift für Physik A Hadrons and Nuclei*, vol. 94, pp. 277–302, 1935.
- [9] D. Jiles and D. Atherton, Theory of Ferromagnetic Hysteresis, *Journal of Magnetism and Magnetic Materials*, vol. 61, no. 1–2, pp. 48 – 60, 1986.
- [10] J. Tellinen, A Simple Scalar Model for Magnetic Hysteresis, *IEEE Transactions on Magnetics*, vol. 24, no. 4, pp. 2200 – 2206, July 1998.
- [11] Soft Magnetic Cobalt-Iron-Alloys, Vacuum-schmelze GmbH, 2001, http://www.vacuum-schmelze.com/fileadmin/docroot/medialib/-documents/broschue-ren/htbrosch/Pht-004_e.pdf (05.21.2012).
- [12] I. Mayergoyz, *Mathematical Models of Hysteresis and their Application*. Elsevier, 2003.

Motor Management of Permanent Magnet Synchronous Machines

Anton Haumer Christian Kral
AIT Austrian Institute of Technology GmbH
Giefinggasse 2, 1210 Vienna, Austria
a.haumer@haumer.at christian.kral@ait.ac.at

Abstract

In this paper the principle of loss and current related motor management of permanent magnet synchronous machines is demonstrated. For this purpose a simplified Modelica model of an interior permanent magnet machine synchronous machine drive is presented. In this model copper, core and friction losses are considered. Simulations then used to determine operating points of minimum current demand and losses, respectively. Based on simulation results some basic insights into motor management are presented. General aspects of motor management modeling are then discussed.

Keywords: Permanent Magnet Synchronous Machine, Field Oriented Control, Optimization of Field Current

1 Introduction

Due to the rising demand on mobility together with contradictions such as climate change and scarce resources a rising variety of electric and hybrid electric vehicles is currently offered. For such vehicles high torque densities and efficiencies of the electric drive are demanded. In particular the total losses of the electric drive shall be as low as possible considering a given derive cycle.

Nowadays, three types of electric machines are commonly used:

- induction machine with squirrel cage
- electrically excited synchronous machines
- permanent magnet (PM) synchronous machine

Typically, asynchronous induction machines are very reliable due to the robust design of the squirrel cage.

However, they need a magnetizing current component to excite the magnetic field. Electrically excited synchronous machines have a separate field winding in the rotor which – for vehicle applications – is usually supplied through slip rings. For induction and electrically excited synchronous machines, additional copper losses arise due to the currents required for exciting magnetic field. In permanent magnet synchronous machines the magnetic field is mainly excited by the permanent magnets. Rare earth magnets have a high energy density and show thus a very high torque and power density.

In the base speed range of either machine, voltage is more or less linearly proportional to speed. Since the voltage is limited by the available battery voltage, higher speeds can only be realized by reducing the magnetic field in the machine – this is the field weakening range. In induction and electrically excited synchronous machines this measure is performed by reducing the field current. In permanent magnet synchronous machines, the permanent magnets cannot be switched off. In order to yet operate the machine in the field weakening range, a current component has to be controlled such way that it counteracts the field caused by the permanent magnet.

For all kinds of machines, one and the same mechanical operating point can be accomplished by different combinations of field and torque generating current components. So obviously, there exists a certain potential of operating an electric drive such way that the total current or losses, respectively, are as low as possible [1, 2, 3]. In this paper the case of a permanent magnet synchronous machine drive is investigated in order to reveal some basic insights on optimal motor management [4].

In particular, the two optimization cases are investigated. First, minimum losses of the machine are examined, since low losses represent a high efficiency of

the machine and thus enable higher energy utilization. Second, minimum current are investigated, since the maximum current is limited by the power electronics and current also influences the total losses of the power converter.

2 Field Oriented Control of PM Machine

The functional principles of induction and synchronous machines are the same: if we feed three sinusoidal currents i_1 , i_2 and i_3 with a time phase shift of 120° to three windings in the stator that are spaced by 120° at the circumference, we achieve a magnetic field wave in the air gap of constant amplitude, rotating with an angular velocity dependent on the frequency of the currents. The rotating magnetic field can be represented by a complex current space phasor,

$$\underline{i} = \frac{2}{3} (i_1 + \underline{a}i_2 + \underline{a}^2i_3) \quad (1)$$

where

$$\underline{a} = e^{j\frac{2\pi}{3}} \quad (2)$$

The zero component

$$i_0 = \frac{1}{3} (i_1 + i_2 + i_3) \quad (3)$$

is usually avoided by normal drive designs since it has no effect on power exchange with the rotor. The current space phasor (1) and the zero component (3) can be interpreted as a linear transformation of the three winding current i_1 , i_2 and i_3 . Rotating the current space phasor (1) into a rotor fixed coordinate frame, it can be represented by current components of the d and q axis,

$$\underline{i}^r = \underline{i}e^{-j\gamma} = i_d + j i_q, \quad (4)$$

see Fig. 1, where γ is the angle between stator and rotor frame. The space phasor transformation can be applied to voltages and flux linkages as well to model the machine behavior. The flux linked with the stator winding can be determined by

$$\underline{\Psi} = \Psi_{PM} + L_{md}i_d + jL_{mq}i_q, \quad (5)$$

see Fig. 2, where the flux of the permanent magnet, Ψ_{PM} , is aligned with the d axis.

The number of pole pairs, p , is defined by the repetition of the stator winding along the circumference. Since the rotor is equipped with a permanent magnet

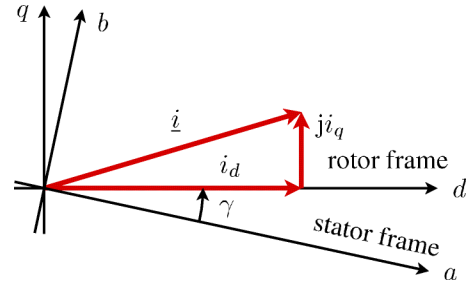


Figure 1: Transformation for the current phasor from the stator to the rotor frame, considering the transformation angle γ

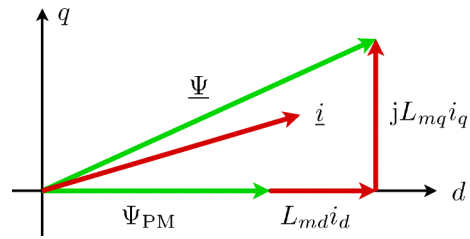


Figure 2: The total stator flux linkage phasor $\underline{\Psi}$ is composed of the flux of the magnet Ψ_{PM} and the inductive components due to the total main inductance and current components

arrangement showing the same number of pole pairs, it is evident that the rotor will try to align in the rotating magnetic field. Thus it is useful to decompose the stator current space phasor into a component aligned with the rotor poles, i_d , and a perpendicular component, i_q (pointing to the pole gap). Having information about the rotor orientation – and therefore about the field orientation – it is possible to control the field current i_d and the torque generating current i_q independent from each other – similar as in DC machines.

2.1 Torque Generation

The electromagnetic torque generated in the air gap of a PM machine is a reaction between magnetic flux linked with the stator winding, $\underline{\Psi}$, and the conjugate complex current space phasor:

$$\tau_{el} = -\frac{3p}{2} \text{Im}(\underline{\Psi} \underline{i}^*) \quad (6)$$

Taking into account the nature of the permanent magnet synchronous machine with different magnetic conductances in the direction of the poles (d -axis) and in direction of the pole gaps (q -axis), we obtain:

$$\tau_{el} = \frac{2p}{3} (\Psi_{PM}i_q + (L_{md} - L_{mq})i_d i_q) \quad (7)$$

In this equation, L_{md} and L_{mq} are the total main inductances in the d and q axis, respectively, representing the magnetic reluctances of these axes. For magnetically isotropic machines with $L_{md} = L_{mq}$ the electromagnetic torque is directly proportional to the product of the magnetic flux linkage of the permanent magnet, Ψ_{PM} , and the current in the q axis. The permeability of permanent magnets is almost equal to air. Thus, magnetically isotropic machines typically have the magnets mounted on the surface of the rotor, see Fig. 3a.

It is evident from (7) that for machines with different magnetic reluctances in d and q axis an additional torque component arises – the reluctance torque. This torque component is proportional to the product of the d and q axis current and the difference between the inductances of the d and q axis. An anisotropic rotor configuration is shown in Fig. 3b interior permanent magnets. In order to gain a higher reluctance torque it is desirable to make the difference between the inductances of the d and q axis as large as possible.

Even though surface mounted permanent magnet synchronous machine reveal a certain potential for minimizing losses [5, 6], the potential is much higher in case of interior mounted permanent magnet synchronous machines [7, 8, 9, 10, 11].

2.2 Losses

In order to minimize current consumption or losses, respectively, the total losses of the PM machine have to be taken into account. For the investigated machine, ohmic losses, core losses and friction losses are considered.

Ohmic losses (copper losses) are directly proportional to the total stator winding resistance, R_s , and the sum of the squared winding currents,

$$P_{Cu} = R_s(i_1^2 + i_2^2 + i_3^2) = \frac{3}{2}R_s \underline{i}_s \underline{i}_s^*. \quad (8)$$

Core losses are usually separated into eddy current losses and hysteresis losses [12, 13]. Some models even take excess losses into account, but these losses are usually inherently considered by the eddy current loss model. In the presented paper machine models of the Modelica Standard Library (MSL) 3.2 are used, so hysteresis losses are not taken into account. The total core losses are thus modeled dependent on the voltage induced by the flux Ψ , linked with the stator winding,

$$P_c = \frac{3}{2}G_c \left(\frac{d\Psi}{dt} \right)^2. \quad (9)$$

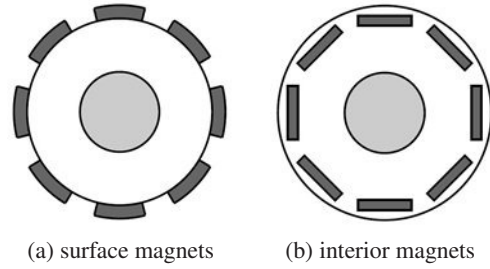


Figure 3: Permanent magnet rotor configurations

Friction torque is modeled as a power of rotor speed – represented by parameter a_f . Friction losses are thus determined by

$$P_f = P_{f,ref} \left(\frac{\Omega}{\Omega_{ref}} \right)^{a_f+1}, \quad (10)$$

where Ω is the mechanical angular rotor speed and index ref indicates a reference point.

Due to the great dependency of torque from the current components i_d and i_q in (7), a high potential for saving current and losses, respectively, is obvious.

2.3 Voltage Induction

The induced voltage under stationary operating conditions is given by

$$\underline{v} = j\omega \underline{\Psi} = j\omega (\Psi_{PM} + L_{md}i_d) - \omega L_{mq}i_q. \quad (11)$$

For zero current in the q axis, the induced voltage solely depends on the flux linkage due to the permanent magnet and the current of the d axis and the electrical angular speed,

$$\omega = \frac{\Omega}{p}. \quad (12)$$

For zero current in both axes the induced voltage rises linearly with speed ω . When the induced voltage exceeds the maximum voltage, determined by the available battery voltage, the field has to be weakened in order to further increase speed. This can be achieved by injecting a negative d axis current component which reduces the total flux linked with the stator winding, see (5).

3 Modelica Model of the Drive

Fig. 4 shows the Modelica model used for investigating the motor management of the drive. A permanent magnet synchronous machine model – taken from the

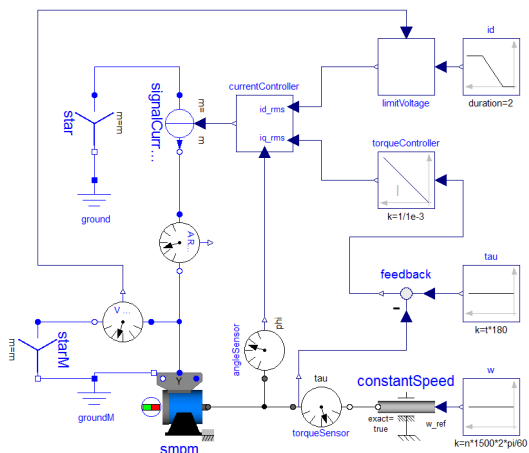


Figure 4: Modelica model of the drive

MSL 3.2 – is fed by a signal current source. This simplification represents an idealized supply case without modeling the details of a power inverter. This way pulse width modulation (PWM) specific effects are not taken into account, since the reference values of the d and q axis current are directly injected into the machine after an inverse space phasor transformation, i.e., calculating the instantaneous three phase currents (block `currentController`).

The shaft of the machine is coupled by an ideal speed source. An angle sensor is used to feed back the angle between stator and rotor frame, γ , to the inverse space phasor transformation.

The field exciting current, i_d , is varied linearly within a given range; the block `limitVoltage` ensures that the actual terminal voltage does not exceed the available DC voltage source, representing the battery voltage of an electric or hybrid electric vehicle. The q current component is determined by an integral controller which is fed by the difference between desired and actual torque. The integral time of this controller is very small such that control specific effects are negligible in the performed investigation.

A certain point of operation as well as the range for optimization are determined by

- torque,
- speed, and
- the range for varying the current component i_d .

Output variables of the investigated model are the total current consumption and the total machine losses. In the presented paper the optimum point of operation is determined manually by either varying speed or

parameter	value	unit
number of pole pairs		
nominal frequency	50	Hz
nominal RMS voltage per phase	100	V
nominal RMS no load voltage per phase	75	V
nominal torque	180	Nm
nominal stator resistance per phase	0.03	Ω
nominal stator stray reactance per phase	0.1	Ω
nominal main reactance per phase, d axis	0.3	Ω
nominal main reactance per phase, q axis	0.6	Ω
nominal core losses	500	W

Table 1: Machine parameters used for the analysis of the motor management

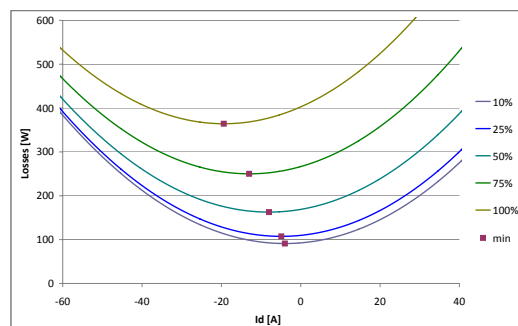


Figure 5: Losses at 25% nominal torque, motor operation, speed variation 10-25-50-75-100% nominal speed

torque, and fixing the remaining parameters and variables, respectively. This way characteristic curves are obtained, see section 4. The machine parameters used for the analysis are presented in Tab. 1.

4 Simulation Results

In this chapter simulation results at different loads, both for motor and generator operation, and different speeds at varying direct axis current are summarized. The optimal d axis currents for minimal machine losses is indicated in the figures.

Fig. 5 shows at 25% nominal torque – motor operation – that machine losses rise with rising speed, due to the increase of core losses. Fig. 6 extends the trend to field weakening. Since only eddy current losses are taken into account, core losses are nearly constant. Decreasing the q axis current demand (limitation of torque proportional to the inverse of speed) decreases copper

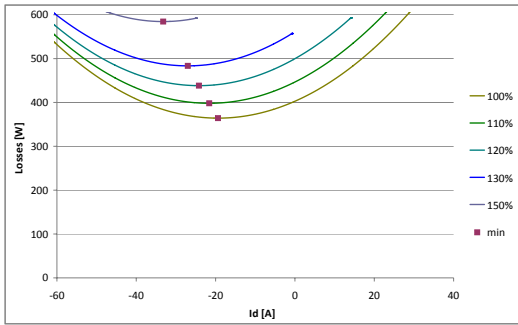


Figure 6: Losses at 25% nominal torque, motor operation, speed variation 100-110-120-130-150% nominal speed

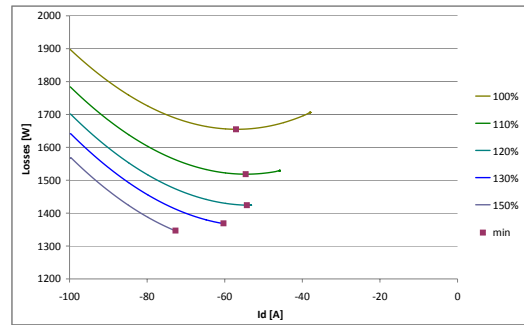


Figure 8: Losses at 100% nominal torque, motor operation, speed variation 100-110-120-130-150% nominal speed

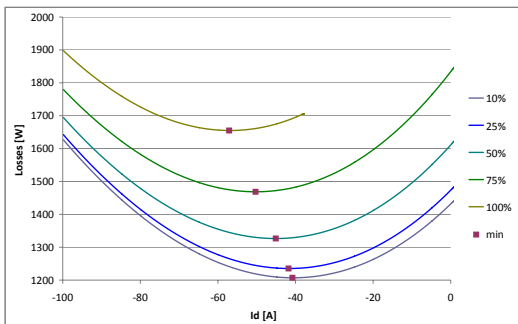


Figure 7: Losses at 100% nominal torque, motor operation, speed variation 10-25-50-75-100% nominal speed

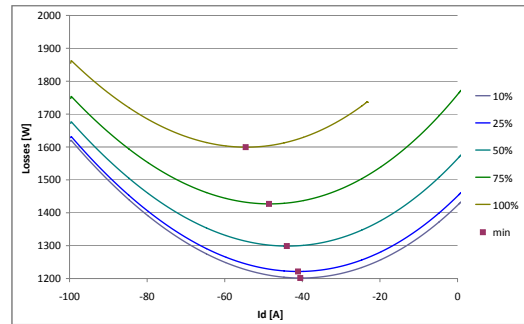


Figure 9: Losses at 100% nominal torque, generator operation, speed variation 10-25-50-75-100% nominal speed

losses, whereas increasing the d axis current – in order to limit the stator voltage – increases copper losses. The trend depends strongly on the actual machine parameters, i.e., inductances and reference losses.

Fig. 7 and Fig. 8 show the same dependencies, but at 100% nominal torque – motor operation. Since for higher torque demand and therefore higher current the influence of copper losses is higher, losses decrease in the field weakening region with rising speed. Additionally it can be observed that a variation of the d axis current is limited by the need of field weakening to avoid exceeding the voltage limit.

Fig. 9 and Fig. 10 depict the same dependencies at 100% nominal torque, but for generator operation, with only small differences compared to motor operation.

Additionally to determining the optimal d axis current for minimum machine losses, minimum total cur-

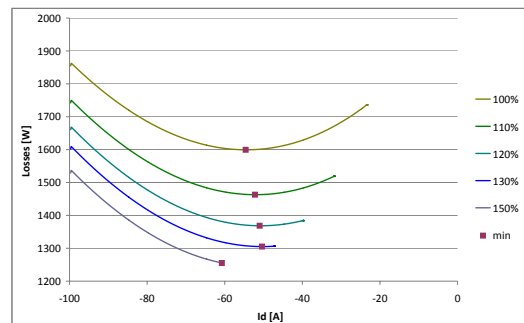


Figure 10: Losses at 100% nominal torque, generator operation, speed variation 100-110-120-130-150% nominal speed

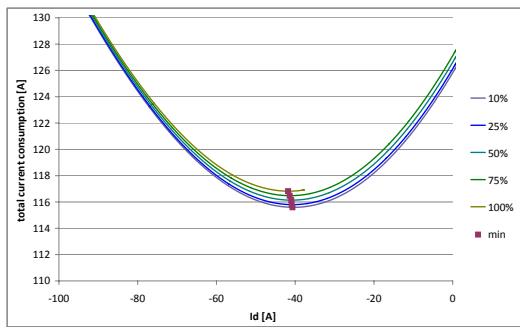


Figure 11: Total current consumption at 100% nominal torque, motor operation, speed variation 10-25-50-75-100% nominal speed

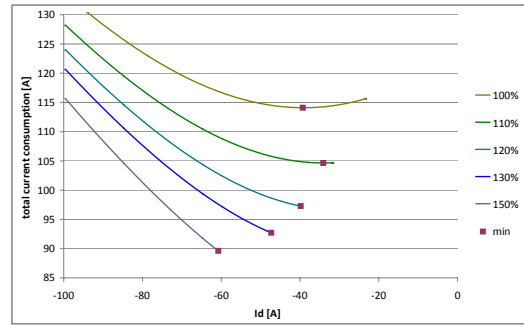


Figure 14: Total current consumption at 100% nominal torque, generator operation, speed variation 100-110-120-130-150% nominal speed

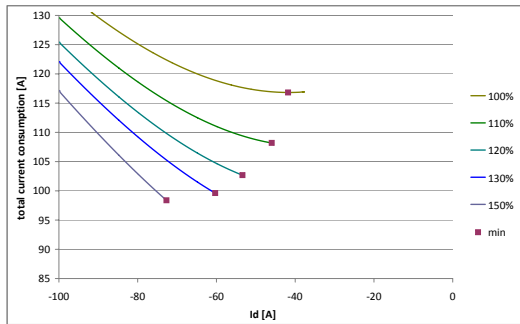


Figure 12: Total current consumption at 100% nominal torque, motor operation, speed variation 100-110-120-130-150% nominal speed

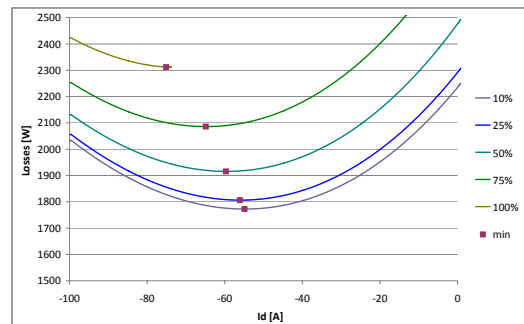


Figure 15: Losses at 125% nominal torque, motor operation, speed variation 10-25-50-75-100% nominal speed

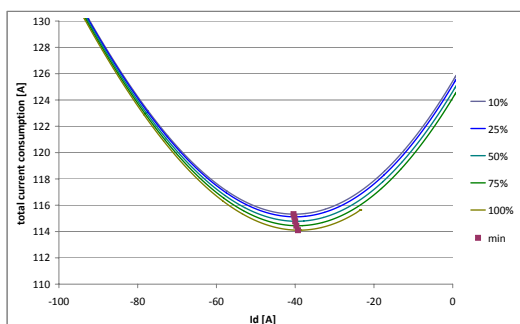


Figure 13: Total current consumption at 100% nominal torque, generator operation, speed variation 10-25-50-75-100% nominal speed

rent is analyzed. The total current consumption for 100% nominal torque in motor operation is depicted in Fig. 11, showing increasing total current for rising speed. This is due to the fact that losses – including rising core losses – have to be covered by electric active power consumption. For the field weakening region – depicted in Fig. 12 – decreasing losses lead to decreasing electric power consumption and therefore decreasing current consumption.

Fig. 13 and Fig. 14 show the same dependencies at 100% nominal torque, but for generator operation. The main difference compared with motor operation results from the fact that core losses cause a braking torque, which reduces the demand for electric torque. In the region of constant magnetic field this leads to decreasing current demand at rising speed.

Fig. 15 shows at 125% nominal torque – overload motor operation – that machine losses rise with rising

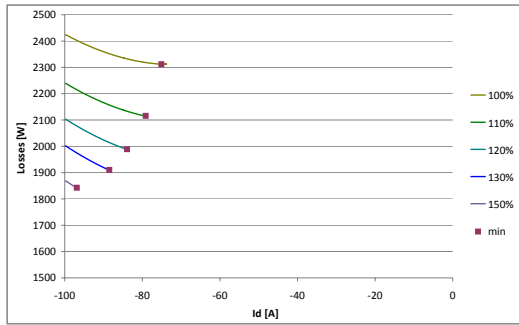


Figure 16: Losses at 125% nominal torque, motor operation, speed variation 100-110-120-130-150% nominal speed

speed, due to the increase of core losses. Fig. 16 extends the trend to field weakening. For speed above nominal speed a high d axis current demand can be noticed. The optimum for each speed can be found at the minimum d axis current that is sufficient to limit stator voltage.

5 Discussion

The presented simulation results rely on a simplified model of a permanent magnet synchronous machine. Based on the obtained results, one could implement an interpolation table, for controlling the optimum d and q axis current in a real application. In this case for a particular speed, torque and available battery voltage, the optimum d and q axis currents have to be pre-calculated and stored in such interpolation table.

However, in a real drive application, some even more complex effects arise which have to be considered properly. In the following the most relevant effects are be discussed:

- The main field inductances are non-linearly dependent on currents due to the saturating characteristic of the core [14]. Additionally, the flux contributions with respect to the d and q axis are not fully magnetically decoupled as suggested in (5). Therefore, cross saturation effects may have to be taken into account [15].
- The ohmic losses are temperature dependent. In order to correctly estimate ohmic losses or the optimal d and q axis currents, temperature has to be either measured or estimated. Temperature, however, complicates setting the optimum operating

point in an online application, since an additional dimension of variability – for picking the optimum d and q axis currents – is added.

- In a real application, the contribution of hysteresis loss may have a significant impact on the exact total core losses. However, this can be accomplished by modifying the core loss equation (9) according to [13].
- In the proposed model, eddy current losses of the permanent magnets are not taken into account. Such losses most likely have to be considered in a real application, sometimes even if the magnets are segmented [16].
- The PWM supply of the power inverter gives rise to certain voltage harmonics which in turn influence the total core losses. In the proposed eddy current model high frequency skin effects with respect to the core flux are not taken into account. However, in particular, PWM related voltage harmonics give rise to additional hysteresis losses due to minor hysteresis loops [17].
- More precisely, in order to maximize the total efficiency of an electric or hybrid electric vehicle, the total losses of the machine and the power converter and the battery have to be minimized, considering all actual current limits and temperatures. In particular the system optimization is a great challenge due the interdependency of the individual losses from the control variables and the (time dependent) limits.

6 Conclusions

The concept of optimizing the field current or the losses of an anisotropic permanent magnet synchronous machine has been demonstrated using a simplified Modelica model. Simulation results have been presented for the base speed and the field weakening region. In the performed investigations the maximum available voltage of the battery is taken into account. Limitations of the presented model are discussed and compared to real drive applications.

References

- [1] S. Morimoto, Y. Tong, Y. Takeda, and T. Hirasa, "Loss minimization control of permanent magnet synchronous motor drives," *Industrial Electronics, IEEE*

- Transactions on*, vol. 41, no. 5, pp. 511–517, oct 1994.
- [2] S. Shinnaka and T. Sagawa, “New optimal current control methods for energy-efficient and wide speed-range operation of hybrid-field synchronous motor,” *Electric Machines and Drives, 2005 IEEE International Conference on*, pp. 535–542, may 2005.
- [3] M. Cao and N. Hoshi, “Electrical loss minimization strategy for interior permanent magnet synchronous motor drives,” *Vehicle Power and Propulsion Conference (VPPC), 2010 IEEE*, pp. 1–6, sept. 2010.
- [4] R. F. Schiferl and T. A. Lipo, “Power capability of salient pole permanent magnet synchronous motors in variable speed drive applications,” *IEEE Transactions on Industry Applications*, vol. 26, no. 1, pp. 115–123, January/February 1990.
- [5] C. Mademlis, J. Xypteras, and N. Margaris, “Loss minimization in surface permanent-magnet synchronous motor drives,” *IEEE Transactions on Industrial Electronics*, vol. 47, no. 1, pp. 115–122, February 2000.
- [6] J.-J. Chen and K.-P. Chin, “Minimum copper loss flux-weakening control of surface mounted permanent magnet synchronous motors,” *IEEE Transactions on Power Electronics*, vol. 18,4, pp. 929–936, 2003.
- [7] C. Mademlis and N. Margaris, “Loss minimization in vector-controlled interior permanent-magnet synchronous motor drives,” *Industrial Electronics, IEEE Transactions on*, vol. 49, no. 6, pp. 1344–1347, dec 2002.
- [8] C. Mademlis, I. Kioskeridis, and N. Margaris, “Optimal efficiency control strategy for interior permanent-magnet synchronous motor drives,” *Energy Conversion, IEEE Transactions on*, vol. 19, no. 4, pp. 715–723, dec. 2004.
- [9] S. Vaez-Zadeh, M. Zamanifar, and J. Soltani, “Non-linear efficiency optimization control of ipm synchronous motor drives with online parameter estimation,” *Power Electronics Specialists Conference, 2006. PESC '06. 37th IEEE*, pp. 1–6, june 2006.
- [10] S. Shinnaka and T. Sagawa, “New optimal current control methods for energy-efficient and wide speed-range operation of hybrid-field synchronous motor,” *IEEE Transactions on Industrial Electronics*, vol. 54, no. 5, pp. 2443–2450, October 2007.
- [11] M. Cao, “Online loss minimization control of ipmsm for electric scooters,” pp. 1388–1392, june 2010.
- [12] C. P. Steinmetz, “On the law of hysteresis,” *Proceedings of the IEEE (reprint of the American Institute of Electrical Engineers Transactions, vol. 9, pp. 3–64, 1892)*, vol. 72, no. 2, pp. 197–222, 1984.
- [13] D. Lin, P. Zhou, W. Fu, Z. Badics, and Z. Cendes, “A dynamic core loss model for soft ferromagnetic and power ferrite materials in transient finite element analysis,” *Conference Proceedings COMPUMAG*, 2003.
- [14] C. Jo, J.-Y. Seol, and I.-J. Ha, “Flux-weakening control of ipm motors with significant effect of magnetic saturation and stator resistance,” *Industrial Electronics, IEEE Transactions on*, vol. 55, no. 3, pp. 1330–1340, march 2008.
- [15] P. Guglielmi, M. Pastorelli, and A. Vagati, “Cross-saturation effects in IPM motors and related impact on sensorless control,” *IEEE Transactions on Industry Applications*, vol. 42, pp. 1516–1522, 2006.
- [16] K. Yamazaki and A. Abe, “Loss analysis of interior permanent magnet motors considering carrier harmonics and magnet eddy currents using 3-d FEM,” *IEEE International Electric Machines & Drives Conference*, vol. 2, pp. 904–909, May 2007.
- [17] Z. Gmyrek, A. Boglietti, and A. Cavagnino, “Estimation and analysis of iron losses in induction motors under sinusoidal and pwm excitation,” *Electrical Machines, 2008. ICEM 2008. 18th International Conference on*, pp. 1–6, sept. 2008.

An approach for modelling quasi-stationary magnetic circuits

Nick Raabe
Sterling Industry Consult GmbH
Lindenstraße 170, 25524 Itzehoe, Germany
nick.raabe@sterlingsihi.de

Abstract

For the design of electrical machines the magnetic circuit has to be modeled. If only the winding layout or the stack length of the motor is changed a complete FEA analysis mostly is not necessary. In this case Modelica is well suited to model the magnetic circuit for quasi-stationary simulations. A new library based on existing standard libraries `MagneticQS` is presented. An induction motor example under no-load conditions shows the basic concept of this library. To enhance and improve the library new models for different types of machines and the possibility of an integral simulation independent from the load conditions is planned.

Keywords: electrical machines; magnetic library; quasi-stationary magnetic circuits

1 Introduction

Up to now the Modelica Standard Library (MSL) contains two packages with different magnetic connectors. Both are subpackages of `Modelica.Magnetic`: `FluxTubes` [1] and `FundamentalWave` [7]. The Modelica concept of providing one potential and one flow variable is implemented here by using the magnetic voltage V_m (A) and the magnetic flux Φ (Vs). The variables in the `FluxTube`-package are of type `Real`. The change of the magnetic flux with respect to time leads to an induced voltage. This package is suitable for all types of transient induction problems. The `FundamentalWave`-package provides the same variables but they are of type `Complex`. These connectors are used for modelling multiphase electric machines in transient operation. The machines presented in this package are identical from the outside behaviour to the machines in `Modelica.Electrical.Machines`. The user has the choice between transforming the electrical stator quantities to space phasors (`Modelica.Electrical.Machines`) or to the magnetic circuit (`Modelica.Ma-`

`gnetic.FundamentalWave`). The equivalence of both models is shown in `Modelica.Magnetic.FundamentalWave.Examples.BasicMachines.AIMC_DOL`.

2 Why another magnetic library?

When designing electric machines the first step is to find a proper geometry. This means to find the best shape for stator and rotor slots, the diameter of the machine and the stack length. Once the winding layout is defined the magnetic circuit can be calculated to determine the magnetizing curve of the machine. This iterative design process means either using a FEA tool or analytical algorithms. After defining the geometry of the machine there are still many options to deviate from this in day-to-day business, e. g. the winding layout can be changed or the quality of the laminations. In this case the FEA mostly is not suitable due to its complexity. This is why Modelica is very helpful to implement a magnetic circuit that is based on algorithms known from the literature but a lot more flexible and clearly described.

These quasi-stationary problems can hardly be simulated with the two existing magnetic packages. Since MSL 3.2 there is the `Modelica.Electrical.QuasiStationary` package available which unfortunately has no connection to the magnetic domain yet. The goal of this paper is to introduce a new magnetic library `MagneticQS` which is similar to the existing ones but takes into account some special requirements for the design of electric machines.

3 Basic concept

The connectors of `MagneticQS` contain complex variables so that they are equal to the connectors of `FundamentalWave`. The difference is that the law of induction is also defined in a complex way. Instead of

saying $v_{ind} \sim d\Phi/dt$ the quasi-stationary representation $v_{ind} \sim j\omega\Phi$ is used. The transformation between electric and magnetic domain is done by the ElectroMagneticConverter. Listings 1 and 2 show the difference of this converter taken from FluxTubes and MagneticQS.

Listing 1: FluxTube converter

```

model ElectroMagneticConverter
SI.Voltage v;
SI.Current i(start = 0,
  stateSelect=StateSelect.prefer);
SI.MagneticPotentialDifference V_m;
SI.MagneticFlux Phi;
parameter Real N(start=1) "Number of
  turns";
equation
  v = p.v - n.v;
  0 = p.i + n.i;
  i = p.i;

  V_m = port_p.V_m - port_n.V_m;
  0 = port_p.Phi + port_n.Phi;
  Phi = port_p.Phi;

  //converter equations:
  V_m = i * N; //Ampere's law
  N * der(Phi) = -v; //Faraday's law
end ElectroMagneticConverter;

```

Listing 2: MagneticQS converter

```

model ElectroMagneticConverter
SI.AngularVelocity omega =
  der(port_p.reference.gamma);
SI.ComplexVoltage v;
SI.ComplexCurrent i;
SI.ComplexMagneticPotentialDifference V_m;
SI.ComplexMagneticFlux Phi;
parameter Real N(start=1) "Number of
  turns";
equation
  v = p.v - n.v;
  Complex(0,0) = p.i + n.i;
  i = p.i;

  V_m = port_p.V_m - port_n.V_m;
  Complex(0,0) = port_p.Phi + port_n.Phi;
  Phi = port_p.Phi;

  //converter equations:
  V_m = i * N; //Ampere's law
  N * j*omega*Phi = -v; //Faraday's law

  //Frequency equations
  Connections.branch(p.reference,
    port_p.reference);
  p.reference.gamma =
    port_p.reference.gamma;
  Connections.branch(n.reference,

```

```

port_n.reference);
n.reference.gamma =
port_n.reference.gamma;
end ElectroMagneticConverter;

```

The main equations are the same for both libraries. The only change is that MagneticQS contains complex variables. The specific characteristic of the Modelica.Electric.QuasiStationary has to be taken into account: The frequency needs also to be considered and transported from one domain to another.

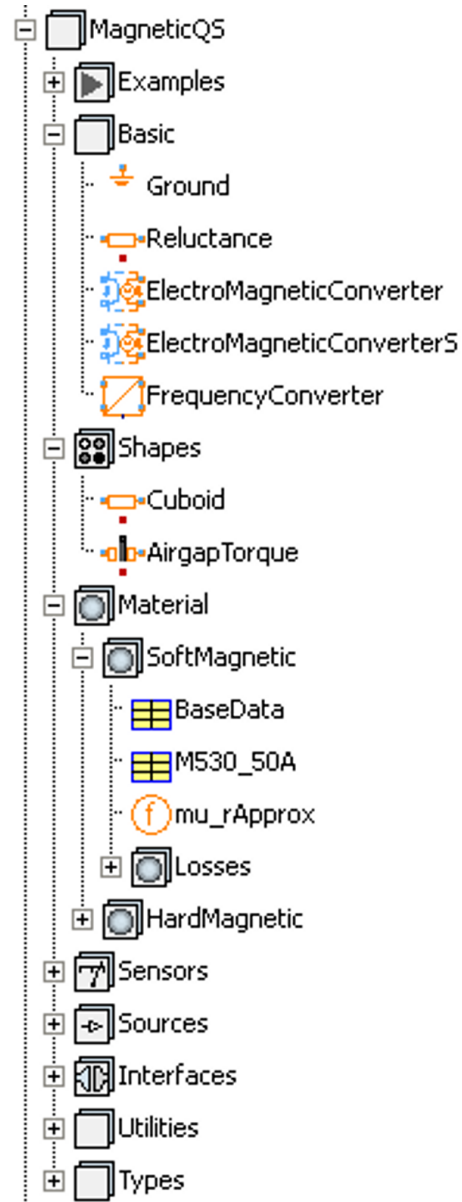


Figure 1: MagneticQS library layout

Once again the differences of the three magnetic libraries (see section 2) should be clarified in the following listing:

- FluxTubes
 - Flux and magnetic voltage are of type Real

- Derivative of flux used in Faraday's law
 - Link to electrical domain via `Modelica.Electric.Analog.Basic`
 - Best choice for transient magnetics (e. g. moving actuators)
- `FundamentalWave`
 - Flux and magnetic voltage are of type `Complex`
 - Derivative of flux used in Faraday's law
 - Link to electrical domain via `Modelica.Electrical.Machines.SpacePhasors`
 - Best choice for space-phasor magnetics (e. g. transient operation of electrical machines)
 - `MagneticQS`
 - Flux and magnetic voltage are of type `Complex`
 - Angular frequency used in Faraday's law (no derivative of flux)
 - Link to electrical domain via `Modelica.Electric.QuasiStationary`
 - Best choice for quasi-stationary magnetics (e. g. magnetic circuits in electrical machines)

4 Complex permeability

As long as ideal reluctances are considered the frequency has no impact on the magnetic flux and potential. The magnetic circuit acts as a coil (in air) and consumes reactive power. For the calculation of iron losses the frequency and the geometry of the flux path haven to be taken in to account. If losses are present the magnetic two-pole not only consumes reactive power but also produces heat (active power). Therefore a heat port is added like it is known from an ohmic resistance in the MSL.

The iron losses consist of two parts: hysteresis and eddy current losses. It is quite common to define the iron losses as the sum of both parts depending on the square of the flux density [5]:

$$P_{fe} = k_{fe} \cdot P_{15} \cdot \left(\frac{f}{50\text{Hz}}\right)^{k_{freq}} \cdot \left(\frac{B}{1\text{T}}\right)^2 m_{fe}. \quad (1)$$

k_{fe} is a correction coefficient that depends on the type of machine (synchronous, asynchronous, DC) and the part of the magnetic circuit (tooth or yoke). P_{15} is listed in standards (e. g. IEC 60404-8-4 [2]) and is also given in material certificates by the lamination

Table 1: Examples for laminations according to IEC 60404-8-4 [2]

Lamination	Thickness mm	P_{15} W/kg	P_{10} W/kg
M270-50A	0.50	2.7	1.1
M400-50A	0.50	4.0	1.7
M800-65A	0.65	8.0	3.6

manufacturer. This value specifies the losses per kg in W and is determined for 50 Hz and 1.5 T. As an alternative also P_{10} (50 Hz, 1 T) can be used. Typical values are given in table 1. If the frequency differs from 50 Hz the exponent k_{freq} (determined empirically) corrects the losses.

In the quasi-stationary domain it is also possible to define the relative permeability as a complex number. Hence the real part describes the magnetic behaviour and the imaginary part describes the losses [3]:

$$\underline{\mu}_r = \mu' - j\mu'' \quad (2)$$

Consequently the reluctance becomes a complex magnetic impedance:

$$\underline{Z}_m = R_m + j\omega L_m. \quad (3)$$

For a cuboid it can be calculated from the geometry (l : length, A : cross section):

$$\underline{Z}_m = \frac{l}{\mu_0 \underline{\mu}_r A}. \quad (4)$$

Thus for the magnetic resistance and the magnetic inductance one can write:

$$R_m = \frac{l}{\mu_0 A} \cdot \frac{\mu'}{\mu_r^2}, \quad (5)$$

$$L_m = \frac{l}{\mu_0 A} \cdot \frac{\mu''}{\mu_r^2}. \quad (6)$$

In analogy to electric circuits the effects of resistance and inductance change: The magnetic resistance R_m leads to reactive power (corresponds with μ'), whereas the magnetic inductance L_m produces losses (corresponds with μ'').

μ' is defined by the approximation function for the magnetizing curve explained in the `FluxTubes`-library. In most cases L_m and μ'' are unknown but P_{fe} is known so that an additional equation based on the power balance is needed to calculate them:

Table 2: Data of example motor [4]

Nominal power	P_n	11 kW
Nominal voltage	U_n	380 V
Nominal frequency	f_n	50 Hz
No. polpairs	p	2
No. stator slots	Z_1	36
No. rotor slots	Z_2	28
Stator winding factor	ξ_1	0.945
No. turns per phase	w_1	168
Stack length	l_{stack}	160 mm

$$L_m = \frac{P_{fe}}{\omega^2 \cdot \Phi^2}. \quad (7)$$

These equations are part of the model `MagneticQS.Shapes.Cuboid`. Up to now only one type of shape is implemented: the cuboid. Since every part of electrical machines (e.g. yoke, tooth) is simplified when modelling magnetic circuits to a rectangular shape this is not a limitation at this early stage of the library. However for further developments other shapes might be useful.

5 Example: Induction machine under no-load condition

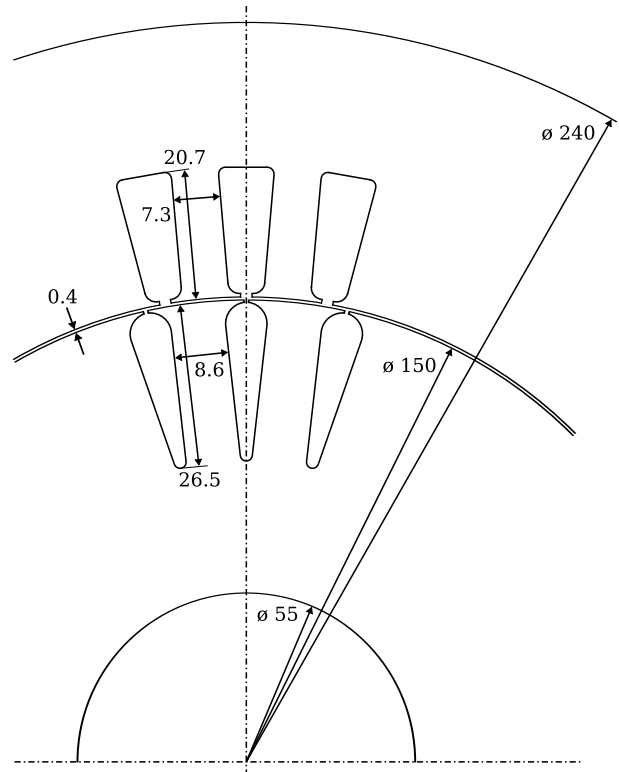
In order to verify the proposed implementation an induction machine is modeled under no-load condition. The motor design is taken from [4]. Table 2 shows the nominal data of the motor, the geometry is given in figure 2.

According to the calculation in [4] the magnetic circuit is divided into five parts:

- stator yoke,
- stator teeth,
- airgap,
- rotor teeth,
- rotor yoke.

For analytical calculations (in contrast to FEA) some special restrictions and simplifications apply:

- The field strength in the stator yoke is strongly nonlinear. Therefore either an additional magnetizing curve for this part of the magnetic circuit is given or a reduction factor [5].
- The flux density in the airgap depends on the width of the stator and rotor slot openings. The airgap length is increased by the so called Carter-Factor.


Figure 2: Stator and rotor geometry [4]

- The magnetic behaviour of the rotor shaft is handled by defining one third of the diameter as iron [4] so that the rotor yoke is enlarged.
- Stray inductances are not taken into account. This approximation is only valid under no-load condition.

The `MagneticQS`-representation is shown in figure 3. Each magnetic impedance is a `MagneticQS.Shapes.Cuboid`. In accordance to [4] only stator yoke and tooth produce losses which is feasible due to the very low frequency in the rotor. In comparison with the calculations in [4] the following deviation has to be mentioned: The book neglects the stator resistance which is quite common when calculating magnetic circuits by hand.

The connection of the electrical and magnetic domain is performed by the model `MagneticQS.Basic.ElectroMagneticConverterS` which is adapted to rotating electrical machine but still based on the converter presented in section 2:

$$V_m = j \cdot I_1 \cdot \frac{3\sqrt{2} \cdot \xi_1 \cdot w_1}{p \cdot \pi}, \quad (8)$$

$$-U_1 = \omega \Phi \cdot \frac{\xi_1 \cdot w_1}{\sqrt{2}}. \quad (9)$$

Table 3 shows the simulation results for the magnetiz-

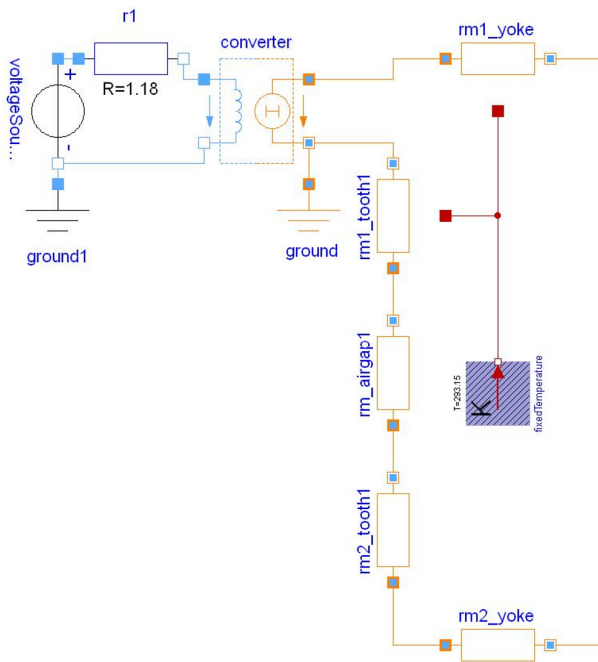


Figure 3: Induction machine no-load example

Table 3: Simulation results (Indices: y: yoke, t: tooth, cu: stator resistance)

Calculation	I_1 A	P_{cu} kW	P_y kW	P_t kW
by hand [4]	5.19	0.00	192.05	85.55
MagneticQS	5.20	31.87	190.97	85.14
MagneticQS, $R_1 = 0$	5.23	0.00	191.88	85.39

ing current and the losses.

When neglecting all losses the example shows the same results as in [4]. By introducing losses in `MagneticQS` slight deviations (see second row of table 3) become present. This proves that the simplifications for hand calculations are valid. The minor influence of the stator resistance is shown in the third row of table 3.

The results show that `MagneticQS` is well suited for the magnetic circuit implementation of electric machines. In comparison with analytical hand calculations it is e. g. no more necessary to calculate the magnetic behaviour and the losses in two steps. `MagneticQS` delivers an integral solution for magnetic circuits in quasi-stationary mode.

6 Summary and future work

This article presents a new magnetic library called `MagneticQS`. It is based on existing libraries but intro-

duces complex variables. The purpose is a clear physical modelling of quasi-stationary magnetic circuits. These are needed in the design phase of electrical machines. The library is designed similar to the existing ones in the MSL. To fulfil the requirements on physical modelling a complex permeability is also introduced. The simulation results show that the new library is well suited to assist the design process for electrical machines. The next step for developing the library is to test different types of machines under load conditions and compare the results with analytical algorithms and FEA. Once this goal is achieved an integral electrical machine magnetic circuit model can be implemented that can be used independently from the state of the machine (no-load, load) which is a great advantage in comparison with existing analytical models. Additional research is needed to find general approaches that eliminate the restriction mentioned in section 5.

References

- [1] Bödrich T. Electromagnetic Actuator Modelling with the Extended Modelica Magnetic Library. Modelica 2008 Conference, Bielefeld, Germany, pp. 221-227, March 3-4, 2008.
- [2] IEC 600404-8-4 Magnetic materials - Part 8-4: Specifications for individual materials - Cold-rolled non-oriented electrical steel sheet and strip delivered in the fully-processed state. 1998.
- [3] Coey J-M-D. Magnetism and Magnetic Materials. 2009.
- [4] Vaske P, Riggert J-H. Elektrische Maschinen und Umformer Teil 2: Berechnung elektrischer Maschinen (*Calculation of electrical machines*). 1974.
- [5] Pyrhönen J, Jokinen T, Hrabovcova V. Design of Rotating Electrical Machines. John Wiley & Sons. 2008.
- [6] Richter R. Elektrische Maschinen Band 1 (*Electrical machines part 1*). 3rd edition. Birkhäuser Verlag. 1967.
- [7] Kral C, Haumer A. The New FundamentalWave Library for Modeling Rotating Electrical Three Phase Machines. Modelica 2011 Conference, Dresden, Germany, March 20-22, 2011.

Functional Mockup Interface 2.0: The Standard for Tool independent Exchange of Simulation Models

T. Blochwitz¹, M. Otter²,
J. Akesson³, M. Arnold⁴, C. Clauß⁵, H. Elmqvist⁶
M. Friedrich⁷, A. Junghanns⁸, J. Mauss⁸, D. Neumerkel⁹, H. Olsson⁶, A. Viel¹⁰

Germany: ¹ITI GmbH, Dresden; ²DLR Oberpfaffenhofen; ⁴University of Halle, ⁵Fraunhofer IIS EAS, Dresden; ⁷SIMPACK, Gilching; ⁸QTronic, Berlin; ⁹Daimler AG, Stuttgart;

Sweden: ⁶Dassault Systèmes, Lund; ³Modelon, Lund;

France: ¹⁰LMS Imagine, Roanne

Abstract

The Functional Mockup Interface (FMI) is a tool independent standard for the exchange of dynamic models and for Co-Simulation. The first version, FMI 1.0, was published in 2010. Already more than 30 tools support FMI 1.0. In this paper an overview about the upcoming version 2.0 of FMI is given that combines the formerly separated interfaces for Model Exchange and Co-Simulation in one standard. Based on the experience on using FMI 1.0, many small details have been improved and new features introduced to ease the use and increase the performance especially for larger models. Additionally, a free FMI compliance checker is available and FMI models from different tools are made available on the web to simplify testing.

Keywords: Simulation; Co-Simulation, Model Exchange; Functional Mockup Interface (FMI); Functional Mockup Unit (FMU);

1 Introduction

The Functional Mockup Interface (FMI) standard version 1.0 (see [1]) was published in 2010 as one result of the ITEA2 project MODELISAR, see Figure 1. In a short time after this first release several modeling and simulation tools started to support FMI. Today, more than 30 tools support FMI 1.0, and it is heavily used in industrial and scientific projects, not only in the automotive sector.

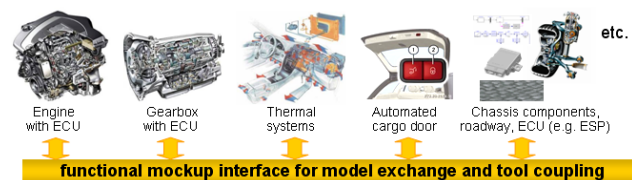


Figure 1: Improving model-based design between OEM and supplier with FMI.

The MODELISAR project ended in Dec. 2011. The maintenance and further development is now performed by the Modelica Association in form of the Modelica Association Project FMI (see <https://www.modelica.org/projects>). FMI was initiated and organized by Daimler AG with the goal to improve the exchange of simulation models between suppliers and OEMs. The further FMI development is performed by 16 companies and research institutes (see Annex). The FMI project is open for FMI interested persons¹ and for (Modelica and non-Modelica) tool vendors supporting FMI.

In this article an overview about the upcoming version 2.0 of FMI is given. This new version combines the formerly separated interfaces for Model Exchange and Co-Simulation in one standard. The specification document was clarified which increases the compatibility of implementations. New features ease the use and increase the performance especially for larger models.

¹ Members of the MA project FMI need not be Modelica Association members, with exception of the project leader.

2 The Functional Mock-Up Interface

2.1 Main Design Ideas

The FMI 2.0 standard consists of two main parts:

1. *FMI for Model Exchange:*
The intention is that a modeling environment can generate C-Code of a dynamic system model in the form of an input/output block, see Figure 2, that can be utilized by other modeling and simulation environments. Models (without solvers) are described by differential, algebraic and discrete equations with time-, state- and step-events.
2. *FMI for Co-Simulation:*
The intention is to couple two or more models with solvers in a co-simulation environment. The data exchange between subsystems is restricted to discrete communication points. In the time between two communication points, the subsys-

tems are solved independently from each other by their individual solver. Master algorithms control the data exchange between subsystems and the synchronization of all slave simulation solvers. The interface allows standard, as well as advanced master algorithms, e.g., the usage of variable communication step sizes, higher order signal extrapolation, and error control.

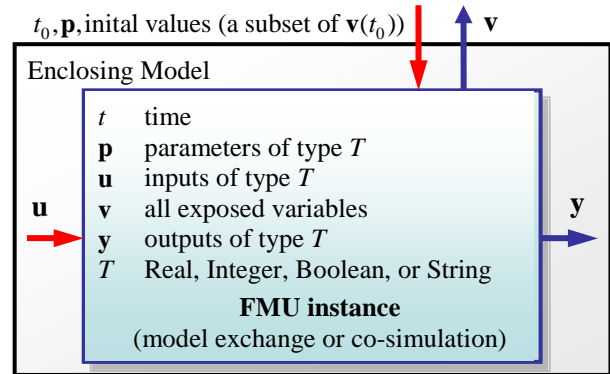


Figure 2: Data flow between the environment and the FMU
Blue/red arrows: Information provided by/to the FMU.

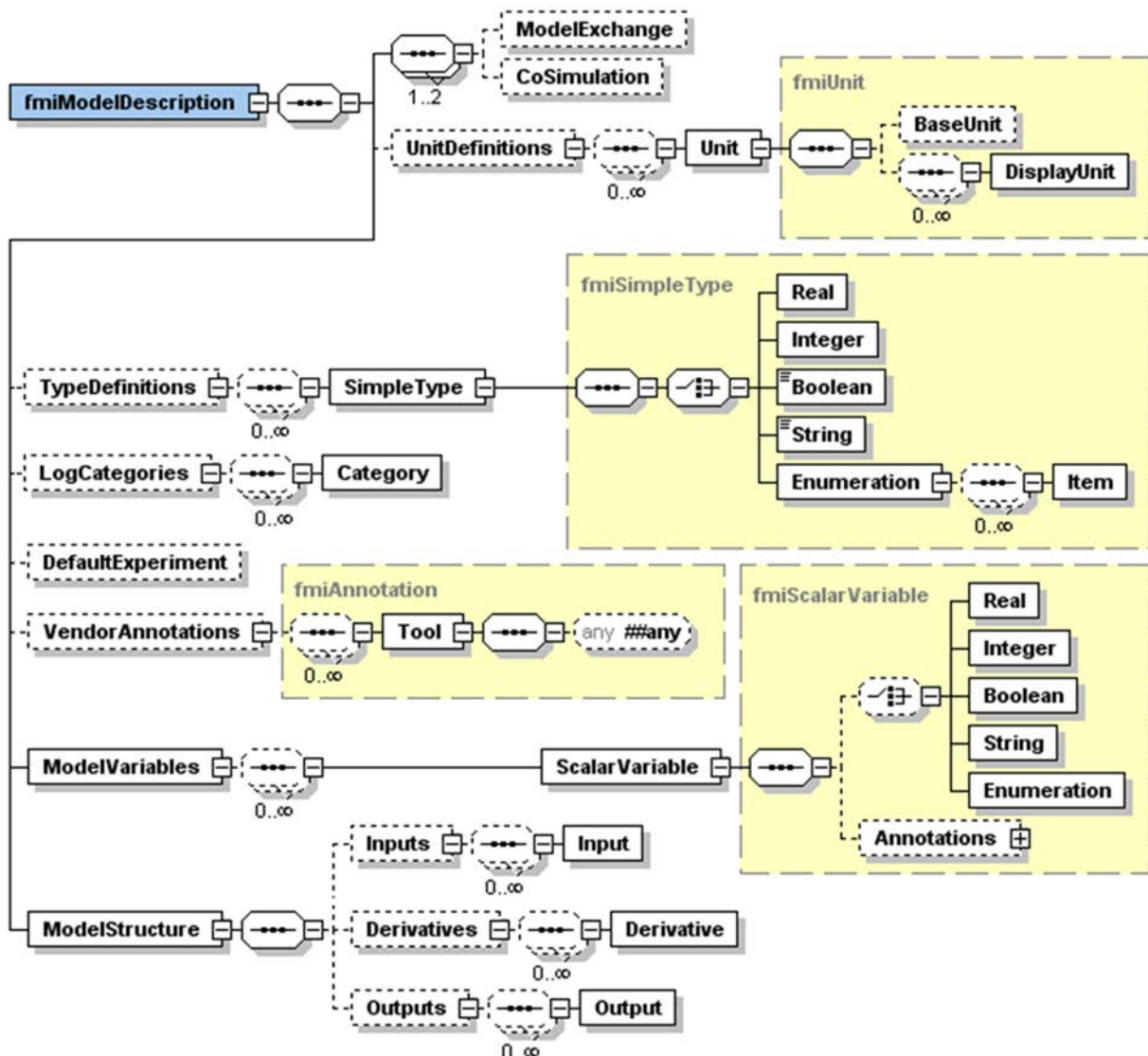


Figure 3: Complete XML schema of upcoming FMI 2.0 (but without attributes and without time synchronization).

2.2 Distribution

A component which implements the FMI is called Functional Mockup Unit (FMU). It consists of one zip-file with extension “.fmu” containing all necessary components to utilize the FMU either for Model Exchange, for Co-Simulation or for both:

1. An *XML-file* contains the definition of all variables of the FMU that are exposed to the environment in which the FMU shall be used, as well as other model information. It is then possible to run the FMU on a target system without this information, i.e., with no unnecessary overhead.
2. A set of *C-functions* is provided to execute model equations for the Model-Exchange case and to setup and run the slaves for the Co-Simulation case. These C-functions can either be provided in source and/or binary form. Binary forms for different platforms can be included in the same model zip-file.
3. Further data can be included in the FMU zip-file, especially a model icon (bitmap file), documentation files, maps and tables needed by the model, and/or all object libraries or DLLs that are utilized.

2.3 Description Schema

All information about a model and a co-simulation setup that is not needed during execution is stored in an XML-file called “modelDescription.XML”. The benefit is that every tool can use its favorite programming language to read this XML-file (e.g., C, C++, C#, Java, Python) and that the overhead, both in terms of memory and simulation efficiency, is reduced. The XML-file is defined by an XML-schema file called “fmiModelDescription.xsd”. In FMI 2.0, the XML-file contains the information both for Model-Exchange and for Co-Simulation.

In Figure 2, the complete XML schema definition is shown. All parts are the same for the two FMI-cases, with exception of the elements “ModelExchange” and “CoSimulation” that contain definitions specific to the respective case. If either one or both of the two elements are present in the XML file, then the respective C-functions are available in the zip-file (usually in binary form as DLL for Windows, and/or as shared object for Linux or Mac). Another essential difference to FMI 1.0 is the new element “ModelStructure” that exposes and provides more details of the model structure.

2.4 C-Interface

The execution interface of FMI 2.0 consists of three header files that define the C-types and –interfaces. The header file “fmiTypesPlatform.h” contains all definitions that depend on the target platform:

```
#define fmiTypesPlatform "standard32"
#define fmiTrue 1
#define fmiFalse 0
#define fmiUndefinedValueReference
        (fmiValueReference)(-1)

typedef void* fmiComponent;
typedef void* fmiComponentEnvironment;
typedef void* fmiFMUState;
typedef unsigned int fmiValueReference;
typedef double fmiReal ;
typedef int fmiInteger;
typedef char fmiBoolean;
typedef const char* fmiString ;
typedef char fmiByte;
```

The underlined, blue type definitions have been newly introduced into FMI 2.0. This header file must be used both by the FMU and by the target simulator. If the target simulator has different definitions in the header file (e.g., “`typedef float fmiReal`” instead of “`typedef double fmiReal`”), then the FMU needs to be re-compiled with the header file used by the target simulator. The header file platform, for which the model was compiled, as well as the version number of the header files, can be inquired in the target simulator with FMI functions.

The type `fmiValueReference` defines a handle for the value of a variable: The handle is unique at least with respect to the corresponding base type (such as `fmiReal`) besides alias variables that can have the same handle. All structured entities, such as records and arrays, are “flattened” into a set of scalar values of type `fmiReal`, `fmiInteger` etc. A `fmiValueReference` references one such scalar. The coding of `fmiValueReference` does not need to be exposed by the modeling environment that generated the model. The data exchange is performed using the functions `fmiSetXXX(...)` and `fmiGetXXX(...)`. XXX stands for one of the types Real, Integer, Boolean, and String. One argument of these functions is an array of `fmiValueReference`, which defines which variables are accessed. The mapping between the FMU variables and the `fmiValueReferences` is stored in the model description XML file.

For simplicity, a “flat” structure of variables is used. Still, the original hierarchical structure of the variables can be retrieved, if a flag is set in the XML-file that a particular convention of the variable

names is used. For example, the Modelica variable name “pipe[3,4].T[14]” defines a variable which is the (3,4) element of an array of records “pipe” of vector T (“.” separates hierarchical levels and “[...]” defines array elements).

Header-file “fmiFunctionTypes.h” contains typedef definitions of all function prototypes of an FMU. When dynamically loading the DLL or shared object of an FMU, these definitions can be used to type-cast the function pointers to the respective function definition. Example for a definition in this header file:

```
typedef fmiStatus fmiSetTimeTYPE
        (fmiComponent, fmiReal);
```

This header file was newly introduced in FMI 2.0 to ease the dynamic loading.

Finally, header file “fmiFunctions.h” contains the function prototypes of an FMU that can be accessed in simulation environments. This header file includes the other two header files from above. Example for a definition in this header file:

```
DllExport fmiSetTimeTYPE fmiSetTime;
```

The goal is that both textual and binary representations of models are supported and that several models using FMI might be present at link time in an executable (e.g., model A may use a model B). For this to be possible the names of the FMI-functions in different models must be different or function pointers must be used. To support the first variant macros are provided in “fmiFunctions.h” to build the actual function names by using a function prefix that depends on how the FMU is shipped. Typically, FMU functions are used as follows:

```
// FMU is shipped with C source code,
// or with static link library
#define FUNCTION_PREFIX MyModel_
#include "fmiFunctions.h"
< usage of the FMU functions >

// FMU is shipped with DLL/SharedObject
#define FUNCTION_PREFIX
#include "fmiFunctions.h"
< usage of the FMU functions >
```

If an FMU is shipped with C source code, or with a static link library, then a function that is defined as “fmiGetReal” is changed by the macros to the actual function name “MyModel_fmiGetReal”. The function prefix is hereby defined in the XML file. A simulation environment can therefore construct the relevant function names by generating code for the actual function call. In case of a static link library, the name of the library is MyModel.lib on Windows, and libMyModel.a on Linux, in other words the function prefix attribute is used as library name.

If an FMU is shipped with a DLL/SharedObject, the constructed function name is “fmiGetReal”, in other words it is not changed. A simulation environment will then dynamically load this library and will explicitly import the function symbols by providing the FMI function names as strings. The name of the library is MyModel.dll on Windows or MyModel.so on Linux, in other words the function prefix attribute is used as library name.

An FMU can be optionally shipped so that it basically contains only the communication to another tool. This is particularly common for co-simulation tasks. In FMI 1.0, the function names are always prefixed with the model name and therefore a DLL/Shared Object has to be generated for every model. FMI 2.0 improves this situation since model names are no longer used as prefix in case of DLL/Shared Objects: Therefore one DLL/Shared Object can be used for all models in case of tool coupling.

3 New Features of FMI 2.0

In this section the main new features introduced by FMI 2.0 are sketched. Note, also many other minor improvements have been introduced, based on the experience in using FMI 1.0. Especially:

- When instantiating an FMU, the simulation environment must report the absolute path to the FMU resource directory also in Model Exchange, in order that the FMU can read all of its resources (for example maps, tables, ...) independently of the "current directory" of the simulation environment where the FMU is used.
- Enumerations have an *arbitrary* (but unique) mapping to integers (in FMI 1.0, the mapping was automatically assigned to 1,2,3,...).
- When enabling logging, log categories can be defined, so that the FMU needs to only generate logs of the defined categories (in FMI 1.0, logs had to be generated for all log categories and they had to be filtered afterwards).
- Explicit alias/antiAlias variable definitions have been removed, to simplify the interface: If variables of the same base type (such as fmiReal) have the same valueReference, they have identical values. A simulation environment may ignore this completely (this was not possible in FMI 1.0), or can utilize this information to more efficiently store results on file.
- Continuous state variables are explicitly listed as FMU variables, and an ordering is introduced for

them, as well as for inputs, and outputs in the XML file, in order that not an (arbitrary) order is selected by the simulation environment. This is essential, for example when linearizing an FMU, or when providing "sparsity" information (see below).

3.1 Unification of FMI for Model Exchange and Co-Simulation

In FMI 1.0 the Model Exchange and Co-Simulation interfaces were defined in two different documents. The XML-description and function definitions were slightly different. In version 2.0 both interfaces are combined in one document and unified. Now one FMU can implement both interfaces at the same time. The presence of the "ModelExchange" or "Co-Simulation" elements in the XML-description indicates which interface is implemented. Which interface is used by the environment is decided by calling the appropriate instantiation function (`fmiInstantiateModel` or `fmiInstantiateSlave`).

In this way the distributed use case (see [1]) which was applicable for Co-Simulation in FMI 1.0 only is supported in the Model Exchange case too. In this use case only the ability of a tool to evaluate the model equations is used, not its solver.

3.2 Classification of Interface Variables

Variables exposed by the FMU are now categorized in a slightly different way in FMI 2.0:

Attribute "**causality**" is an enumeration that defines the causality of the variable. Allowed values are:

- **parameter**: An *independent* variable that must be constant during simulation.
- **input**: The variable value can be provided from another model.
- **output**: The variable value can be used by another model. The algebraic relationship to the inputs is defined in element *ModelStructure*.
- **local**: Local variable that is calculated from other variables. It is not allowed to use the variable value in another model

Attribute "**variability**" is an enumeration that defines the time dependency of the variable, in other words it defines the time instants when a variable can change its value. Allowed values are:

- **constant**: The value of the variable never changes.
- **fixed**: The value of the variable is fixed after initialization.
- **tunable**: The value of the variable is constant between externally triggered events due to

changing variables with causality = "parameter" or "input" (see explanation below).

- **discrete**: The value of the variable is constant between internal events (= time, state, step events defined implicitly in the FMU).
- **continuous**: No restrictions on value changes.

The new value "**tunable**" introduced in FMI 2.0 allows a modeling environment to expose independent parameters that can be manually "tuned" during simulation (for example, during simulation a modeler might change the gain of a PID controller, or the load mass of a drive train in order to quickly improve the design).

"Tuning a parameter" during simulation does not mean to "change the parameter online" during simulation (since this might introduce Dirac impulses). Instead, this is a short hand notation for:

1. Stop the simulation at an event instant (usually, a step event, in other words after a successful integration step).
2. Change the values of the tunable parameters.
3. Compute all parameters that depend on the tunable parameters.
4. Resume the simulation using as initial values the current values of all variables and the new values of the parameters.

With this interpretation, changing parameters online is "clean", as long as these changes appear at an event instant.

3.3 Save and Restore of FMU state

An FMU has an internal state consisting of all values that are needed to continue a simulation. This internal state consists especially of the values of the continuous states, discrete states, iteration variables, parameter values, input values, file identifiers and FMU internal status information. With newly introduced (optional) functions, the internal FMU state can be copied and the pointer to this copy is returned to the environment. The FMU state copy can be set as current FMU state, in order to continue the simulation from it. This feature introduced in FMI 2.0 can be for example used:

- For iterative *co-simulation master algorithms* (get the FMU state for every accepted communication step; if the follow-up step is not accepted, restart co-simulation from this FMU state).
- For *nonlinear Kalman filters* (get the FMU state just before initialization; in every sample period, set new continuous states from the Kalman filter algorithm based on measured values; integrate to the next sample instant and inquire the predicted

continuous states that are used in the Kalman filter algorithm as basis to set new continuous states).

- For *nonlinear model predictive control* (get the FMU state just before initialization; in every sample period, set new continuous states from an observer, initialize and get the FMU state after initialization. From this state, perform many simulations that are restarted after the initialization with new input signals proposed by the optimizer).

Furthermore, the FMU state can be serialized and copied into a byte vector. This can, for example be used to perform an expensive steady-state initialization, copy the received FMU state in a byte vector and store this vector on file. Whenever needed, the byte vector can be loaded from file, can be deserialized and the simulation can be restarted from this FMU state, in other words from the steady-state initialization.

3.4 Dependency Information

In FMI 1.0 only the dependencies of outputs on inputs could be defined by the element “DirectDependency” in the XML-description. In FMI 2.0 this information and the dependencies of outputs w.r.t. state variable and of derivatives w.r.t. inputs and state variables can be provided using the element “*ModelStructure*”. Under this element ordered lists of inputs, derivatives (with their associated state variable names) and outputs are provided. At each output and derivative additional attributes define the dependency on inputs and state variables. Not only the dependency itself but also the kind of dependency is defined here. It can be indicated whether the dependency is *nonlinear*, *fixed* (the dependency is linear, the factor is constant after initialization) or *discrete* (the factor might change after events). Using this information a tool can decide at which stage of the solution process the respective entries of the Jacobian matrices are to be retrieved.

The dependency information of outputs can be utilized for detection of algebraic loops when FMUs are connected with other parts of a model. In addition to that dependency information is necessary for usage of sparse matrix techniques on Jacobian matrices.

Assume for example that the following equations are defined:

$$\frac{d}{dt} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} f_1(x_2) \\ f_2(x_1) + 3 \cdot p^2 \cdot x_2 + 2 \cdot u_1 + 3 \cdot u_3 \\ f_3(x_1, x_3, u_1, u_2, u_3) \end{bmatrix}$$

$$y = g_1(x_2, x_3)$$

where u_1 and u_2 are continuous-time inputs (variability=“continuous”), u_3 is a discrete-time input (variability=“discrete”), and p is a fixed parameter (variability=“fixed”). The structure of these equations can then be defined optionally in the following way in the XML file:

```
<ModelStructure>
  <Inputs>
    <Input name="u1" />
    <Input name="u2" />
    <Input name="u3" />
  </Inputs>

  <Derivatives>
    <Derivative name="der(x1)" state="x1"
      stateDependencies="2"
      inputDependencies=" " />
    <Derivative name="der(x2)" state="x2"
      stateDependencies="1 2"
      stateFactorTypes = "nonlinear fixed"
      inputDependencies="1 3"
      inputFactorTypes = "fixed fixed" />
    <Derivative name="der(x3)" state="x3"
      stateDependencies="1 3" />
  </Derivatives>

  <Outputs>
    <Output name="y"
      stateDependencies="2 3"
      inputDependencies=" " />
  </Outputs>
</ModelStructure>
```

3.5 Jacobian Matrices

Partial derivatives of FMU variables with respect to inputs or state variables (Jacobian matrices) are needed for implicit integration methods, for linearization of FMUs, or for usage in extended Kalman filters. Especially for large models the numerical computation of Jacobian matrices is time consuming. For that reason FMUs can optionally provide functions to retrieve partial derivatives (complete Jacobians) or directional derivatives of some variables w.r.t. some others.

The sparsity pattern defined under “*ModelStructure*” (see section above) can be utilized for efficient data storage and matrix operations on sparse Jacobians. FMI does not define a specific storage schema. The calling environment is free to use its own schema by the following approach. The environment has to provide a function pointer to a call back function `setMatrix` as argument of `fmiGetPartialDe-`

rivatives. The FMU calls this function to set respective matrix elements.

The FMU internally is free to use efficient numerical methods for Jacobian computation, use a symbolically deduced algorithm or automatic differentiation.

3.6 Precise Time Event Handling

The details of precise time event handling in FMI were still under discussion before the editorial deadline of this paper. Hence we cannot present a detailed description here. The development work is complicated since several aspects have to be considered:

- The synchronous features of Modelica 3.3 [2] should be supported.
- FMI should also be useable by tools that do not support synchronous time event handling.
- The time event handling is to be defined in a way that allows backward compatible extensions.

3.7 Improved Unit Definitions

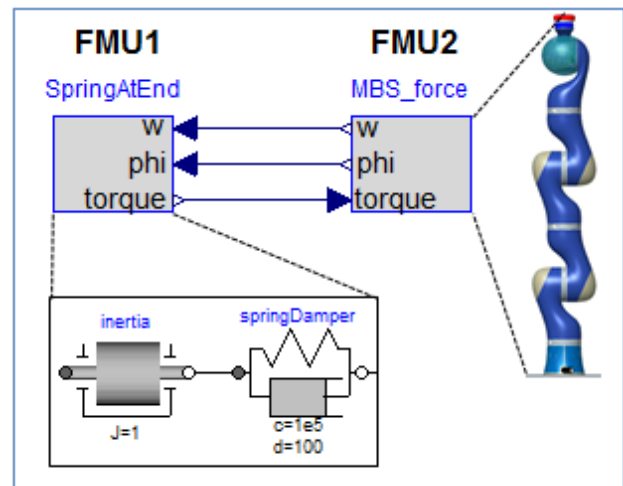
The unit definitions have been improved in FMI 2.0: The tool-specific unit-name can optionally be expressed as function of the 7 SI base units and the SI derived unit “rad”. It is then possible to check units when FMUs are connected together (without standardizing unit names as needed in FMI 1.0), or to convert variable values that are provided in different units (for the same physical quantity). In the specification it is sketched how to utilize this information for connection checks, dimensional checks, or unit propagation. The trick is to treat the derived unit “rad” either as “rad” (for connection checks and unit propagation) or as “1” (for dimensional checks) depending on the situation.

4 Examples

In this section two examples are shown that demonstrate the structure of the XML file and especially how FMUs can be connected together. The use case is an often occurring situation where two FMUs shall be connected that have a mechanical interface.

4.1 FMU as Force Element

In the first example, FMU 1 consists of a one-dimensional rotational drive train with an inertia that is connected to a rotational spring/damper system and the end point of the spring/damper system shall be used as interface of this FMU, see next figure:



In multi-body system terminology, this is called a “force element”. Typically, FMU 1 would be a complicated device, e.g., a controlled electrical motor with a gearbox, but the essential part is the force element at the interface. The inputs to FMU 1 are the angle ϕ and the angular velocity w of the end point of the spring/damper system. The output would be the torque generated by the spring/damper. It is calculated with the simple equation

$$\text{torque} = c \cdot (\phi - \text{inertia}.\phi) + d \cdot (w - \text{inertia}.w)$$

where c is the spring and d is the damper constant.

This FMU is then connected to a multi-body system FMU, for example a robot, and drives a revolute joint. The FMU 2 provides ϕ and w as output (from the relative joint coordinates) and gets the torque as input.

The XML-file of FMU 1 has the following structure:

```
<?XML version="1.0" encoding="UTF-8"?>
<fmiModelDescription
  xmlns:xsi="http://www.w3.org/2001/.."
  xsi:noNamespaceSchemaLocation="fmiModel.."
  fmiVersion="2.0"
  modelName="FMU_Coupling.DriveTrain_TorqueAtEnd"
  guid="{a4976b5c-b9f7-432a-9dd3-e80bafaac060}"
  generationTool="..."
  generationDateAndTime="2012-07-15T12:52:13Z"
  variableNamingConvention="structured"
  numberOfEventIndicators="0">
  <ModelExchange
    modelIdentifier="FMU_0Coupling_..."
    canGetAndSetFMUstate="true"
    providesPartialDerivativesOf_Derivative
      Function_wrt_States="true"
    ...
    providesDirectionalDerivatives="true"/>
  <CoSimulation
    modelIdentifier="FMU_0Coupling_..."
    canHandleVariableCommunicationStepSize="true"
    canHandleEvents="true"
    canInterpolateInputs="true"
    canSignalEvents="true"
    canGetAndSetFMUstate="true"
    .../>
```

```

<UnitDefinitions>
  <Unit name="N.m">
    <BaseUnit kg="1" m="2" s="-2"/> </Unit>
  </UnitDefinitions>

<TypeDefinitions>
  <SimpleType
    name="Modelica.SIunits.Torque">
    <Real quantity="Torque" unit="N.m"/>
  </SimpleType>
  ...
</TypeDefinitions>

<DefaultExperiment startTime="0.0"
  stopTime="1.0" tolerance="0.0001"/>

<ModelVariables>
  <ScalarVariable
    name="torque"
    valueReference="335544320"
    description="Torque in flange"
    causality="output">
    <Real
      declaredType=
        "Modelica.Blocks.Interfaces.RealOutput"
      unit="N.m"/>
    ...
  </ScalarVariable>
</ModelVariables>

<ModelStructure>
  <Inputs>
    <Input name="phi"/>
    <Input name="w" derivative="1"/>
  </Inputs>
  <Derivatives>
    <Derivative
      name="der(inertia.phi)"
      state="inertia.phi"
      stateDependencies="2"
      inputDependencies=""/>
    <Derivative
      name="der(inertia.w)"
      state="inertia.w"/>
  </Derivatives>
  <Outputs>
    <Output name="torque"
      inputDependencies="1 2"
      inputFactorKinds="fixed fixed"/>
  </Outputs>
</ModelStructure>
</fmiModelDescription>
    
```

Most of the elements should be self-explanatory. The interesting part for the connection is element “ModelStructure” at the end. Output torque depends on the first and the second input, i.e. on phi and w. Furthermore, the attributes fixed define that the inputs enter the equation for the output with fixed linear factors:

$$\text{torque} = p_1 \cdot \text{phi} + p_2 \cdot w + f(\dots)$$

where p_1 and p_2 are constants that are fixed after initialization. Additionally, for input w the attribute derivative = “1” is defined. The meaning is that w is the derivative of the first input, i.e. of phi. This derivative information for inputs and outputs is essential in order that a coupling tool can check that an input is really the derivatives of another input by checking the derivative attributes of the outputs from another FMU.

The XML-file for FMU 2 looks similar. We will concentrate only on the ModelStructure element:

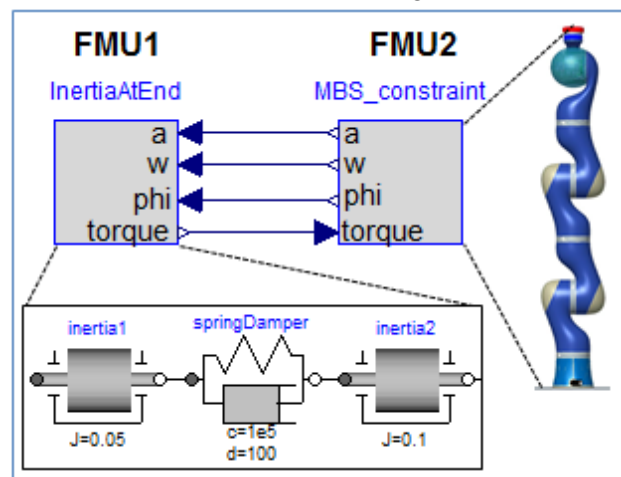
```

<ModelStructure>
  <Inputs>
    <Input name="torque"/>
  </Inputs>
  <Derivatives>
    ...
  <Outputs>
    <Output
      name="phi"
      stateDependencies="1"
      inputDependencies=""/>
    <Output
      name="w"
      derivative="1"
      stateDependencies="2"
      inputDependencies=""/>
  </Outputs>
</ModelStructure>
    
```

The important point is that empty inputDependencies lists are defined for the outputs. This means that the outputs phi and w do not directly depend on the input torque. As a result, when connecting FMU 2 to FMU 1, the outputs phi and w are provided by FMU 2. FMU 1 computes its output torque that is an input to FMU 2. Since the FMU 2 outputs do not depend on this input, there is no algebraic loop and the computation is simple.

4.2 FMUs with Coupling Constraint

The second example is the more often occurring case, but is more involved. FMU 1 is again a one-dimensional rotational drive train, but ends this time with a rotational inertia, see next figure:



Since FMU 1 is connected to a joint of FMU 2, the coupling leads to a constraint equation that states that the angle of the revolute joint of FMU 2 is identical to the angle of inertia2 in FMU 1. It is well-known that such a model cannot be transformed by purely algebraic transformations into a state space

form (this is a so called higher index system²), and that the first and second derivatives of this constraint equation is needed. For this reason, FMU 2 provides the angle `phi` of the revolute joint, its first derivative `w` (the angular velocity) as well as its second derivative `a` (the angular acceleration) to FMU 1. In turn FMU 1 provides the reaction torque to FMU 2. The “ModelStructure” elements of the two FMUs have now the following structure:

FMU 1:

```
<ModelStructure>
  <Inputs>
    <Input name="phi"/>
    <Input name="w" derivative="1"/>
    <Input name="a" derivative="2"/>
  </Inputs>
  <Derivatives>
    ...
  <Outputs>
    <Output
      name="torque"
      inputDependencies="3"
      inputFactorKinds="fixed"/>
  </Outputs>
</ModelStructure>
```

FMU 2:

```
<ModelStructure>
  <Inputs>
    <Input name="torque"/>
  </Inputs>
  <Derivatives>
    ...
  <Outputs>
    <Output
      name="phi"
      stateDependencies="1"
      inputDependencies=""/>
    <Output
      name="w"
      derivative="1"
      stateDependencies="2"
      inputDependencies=""/>
    <Output
      name="a"
      derivative="2"
      stateDependencies="1"/>
  </Outputs>
</ModelStructure>
```

The ModelStructure of FMU 1 states that its output `torque` depends on its 3rd input `a` and that `a` enters with a fixed factor. Therefore, the following equation is present:

$$\text{torque} = J \cdot a + f1(\text{<states>})$$

where J is a constant quantity that is fixed after initialization (this is the inertia of component `inertia2`) and `f1(..)` is an additional functional dependency of the states of the FMU, but not of its inputs.

² Simulating such a higher index system of index 3 directly will usually fail with an error message of the integrator that there is no convergence.

The ModelStructure of FMU 2 states that its 3rd output `a` depends on all of its inputs, i.e. on `torque` (since no `inputDependencies` attribute is defined):

$$a = f2(\text{torque}, \text{<states>})$$

Therefore, when the two FMUs are connected together an algebraic loop in the angular acceleration `a` and in the reaction `torque` appears. The environment has therefore to either use a differential-algebraic equation solver, or has to solve a nonlinear algebraic loop over the two FMUs. The latter case can be improved by using Jacobian information:

As will be explained below, it is possible to compute the factor J once after initialization and the term `f1` at every model evaluation (which turns out to be a cheap operation for a drive train). It is then only necessary to solve a nonlinear algebraic loop over FMU 2 and the simple equation of FMU 1. Additionally, the Jacobian of the FMU 2 equation can be computed. Since for all mechanical systems the FMU 2 equation depends linearly on the unknowns, a nonlinear solver will converge with the provided Jacobians within one step.

An often occurring situation is that FMU 1 is imported into a multi-body program and coupled to a joint. In such a case, the multi-body code gets the information about the linear equation of FMU 1. Since the multi-body program has to solve a linear equation system in the accelerations and in the forces/torques of its mechanical system, just the simple linear equation of FMU 1 has to be added and in every model evaluation only one linear equation system has to be solved.

To summarize, the coupling in this example becomes more complicated and linear or non-linear equation systems have to be solved. This is relatively cheap provided the information about linear dependencies and/or Jacobians are utilized.

The partial derivatives of output variables with respect to input variables can be computed with function `fmiGetDirectionalDerivative`. For the case of one output variable y as function of states \mathbf{x} and of one input u , this function assumes an equation of the form:

$$y = g(\mathbf{x}(t), u(t), t)$$

The function calculates:

$$\Delta y = \frac{\partial g}{\partial u} \Delta u$$

where the seed Δu is given as an explicit input argument. Therefore, calling `fmiGetDirectionalDerivative` for the output `torque` with respect to input `a` and with $\Delta a=1$, the function will return the partial derivative, that is J . The value of `f1` is computed by providing an input `a=0` and computing the output `torque`, that is `torque = f1(<states>)`. Similari-

ly, the partial derivative of the FMU 2 equation can be computed.

As a final remark: When FMU 1 is modeled in Modelica, then the derivative relationships between the inputs of the FMU must be defined, otherwise a Modelica translator cannot process the model. There is no direct Modelica language element available to define this. However, with component Modelica.Mechanics.Rotational.Sources.Move from the Modelica Standard library this relationship is expressed (based on language elements to express that a function is a derivative of another function).

5 Increasing Quality of FMI Implementations

The FMI project provides an infrastructure to increase the quality and compatibility of implementations in different tools. A repository of FMUs generated by different tools and reference results are publicly available at the svn server:

https://svn.fmi-standard.org/fmi/trunk/Test_FMUs

In this way tool vendors are able to cross check their implementations in an easy way. We hereby would like to ask tool vendors that export FMUs, to provide FMUs of their tools by sending an email with the FMUs to info@fmi-standard.org.

Additionally, the Modelica Association contracted the development of an open source FMI compliance checker. This tool is now available for FMI 1.0 in source code, and as executable for Windows and Linux under the svn address from above. It will be available for FMI 2.0 soon after FMI 2.0 is released.

6 FMI Usage

FMI is used in industrial and scientific projects by several companies and research institutions:

In all new gearbox projects for Mercedes-Benz passenger cars FMI is used for software-in-the-loop simulations [3]. Control software and FMUs coming from different modeling environments run in closed-loop in the virtual ECU tool Silver on Windows PC in order to validate, test and debug control software.

Before FMI, vehicle models had to be imported through various vendor and version specific import procedures into Silver. This was expensive and error prone. Thanks to the FMI, these bridges have now been replaced by a uniform import interface, increasing thereby the cost-benefit ratio of simulation in this domain.

In mechatronic gearshift simulations for commercial vehicles at Daimler AG FMI is utilized twice [4]. At first controller software is connected to a detailed 1D powertrain model in SimulationX. Afterwards this model is exported as FMU and imported to the multibody system simulation tool Simpack. There it is connected to a detailed truck model. This allows the holistic simulation and optimization of the shifting comfort.

At IFP Energies Nouvelles, FMI for Model Exchange is used to parallelize the execution of complex internal combustion engine models in the tool xMOD (see [5]). The models have around 100 - 300 state variables, with integration step-sizes that can reach some microseconds. Their use is mainly intended to validate engine controls. The final target is to enable the execution in real-time, for hardware in the loop simulations.

In [6], an algorithm is implemented for derivative-free optimization implemented in Python and applied to parameter optimization of FMUs is introduced. The FMUs are loaded and simulated using the PyFMI package (<http://www.pyfmi.org>). The optimization algorithm is applied to a Volvo truck engine to identify model parameters based on measurement data from a test cycle.

In [7] the FMI based co-simulation master from Fraunhofer is used to develop, implement and test sophisticated algorithms for the co-simulation of FMUs generated by Dymola.

Dassault Systèmes uses FMI for academically trainings. Student teams work with CATIA V6 and define both a 3D CATIA representation of a NXT robot as well as the controller software. Practically, the real robot has sensors and actuators and is piloted from a smartphone remote command, while the FMU based logical control is executed in a CATIA session. All these items are FMI and Bluetooth connected.

The solution has been delivered to Georgia Institute of Technology and University of Detroit Mercy (US High Schools), also related to a cooperation with Ford Motors Foundation.

In the field of modeling and simulation of building energy systems FMI is also used. In [8] FMI is utilized to connect a building model with a Modelica model of the heating system.

In 2012, the International Energy Agency, under the implementing agreement on Energy Conservation in Buildings and Community Systems, approved the five-year Annex 60 proposal "New generation computational tools for building and community energy systems based on the Modelica and Functional Mockup Interface standards." Eleven countries are expected to participate in sharing, developing and

deploying free open-source contributions for modeling and simulation of energy systems of buildings and communities, based on Modelica and Functional Mockup Interface standard.

The Lawrence Berkeley National Laboratory (LBNL) released an FMI for co-simulation import interface in version 7.1 of the EnergyPlus building simulation program. Work is also in progress to export EnergyPlus as a FMU for Co-Simulation. UC Berkeley and LBNL have been developing JFMI, a Java Wrapper for FMI for Co-Simulation and Model Exchange. JFMI will be used to integrate an FMI import interface in Ptolemy II, a software environment for design and analysis of heterogeneous systems.

The Institute for the Sustainable Performance of Buildings has been developing a web-based eLearning tool, Learn Green Buildings (<http://learngreenbuildings.org>), in which a Web interface communicates with an FMU for Co-Simulation that computes the dynamic response of building energy and control systems. The tool will allow students to interactively operate a simulated, realistic building system, to test energy-saving measures and to explore the effects of faults in equipment and controls.

7 Conclusions and Outlook

FMI is an established standard for Model Exchange and Co-Simulation. The upcoming version 2.0 improves the compatibility of implementations by a clarified specification. New features increase usability and performance especially for large models.

This version will be stable for the next years. If necessary, minor backwards compatible releases will be available to improve and clarify the specification and to support new features. Current development tasks are the exchange of structured data and arrays of variable size and support of the new synchronous features of the Modelica language [2].

The further development of FMI is organized under the hood of the Modelica Association. The FMI Modelica Association Project is of course open for non Modelica tool vendors and organizations. From the 16 members of the FMI Steering Committee and Advisory Group, only five are Modelica Tool vendors.

Companies and organizations which are interested to contribute to FMI development or request features are invited to contact the FMI project via info@fmi-standard.org.

8 Acknowledgements

The authors wish to thank all the contributors to the FMI specification (see Annex).

Parts of this work were supported by the German BMBF (Förderkennzeichen: 01IS08002), the French DGCIS, and the Swedish VINNOVA (funding number: 2008-02291) within the ITEA2 MODELISAR project (<http://www.itea2.org/project/result/download/result/5533>) The authors appreciate the partial funding of this work.

9 References

- [1] T. Blochwitz, M. Otter, M. Arnold, C. Bausch, C. Clauß, H. Elmqvist, A. Junghanns, J. Mauss, M. Monteiro, T. Neidhold, D. Neumerkel, H. Olsson, J.-V. Peetz, S. Wolf: **The Functional Mockup Interface for Tool independent Exchange of Simulation Models**. 8th International Modelica Conference. Dresden 2011. Download: <http://www.ep.liu.se/ecp/063/013/ecp11063013.pdf>
- [2] Modelica Association: **Modelica – A Unified Object-Oriented Language for Systems Modeling. Language Specification, Version 3.3**. May 9, 2012.
- [3] E. Chrisofakis, A. Junghanns, C. Kehrer, A. Rink: **Simulation-based development of automotive control software with Modelica**. 8th International Modelica Conference. Dresden 2011. Download: <http://www.ep.liu.se/ecp/063/001/ecp11063001.pdf>
- [4] A. Abel, T. Blochwitz, A. Eichberger, P. Hamann, U. Rein: **Functional Mock-up Interface in Mechatronic Gearshift Simulation for Commercial Vehicles**. 9th International Modelica Conference. Munich, 2012.
- [5] Abir Ben Khaled, Mongi Ben Gaid, D. Simon, G. Font: **Multicore simulation of powertrains using weakly synchronized model partitioning**. Accepted for 2012 IFAC Workshop on Engine and Powertrain Control, Simulation and Modeling. Rueil-Malmaison, 2012
- [6] S. Gedda, C. Andersson, J. Åkesson, S. Diehl: **Derivative-free Parameter Optimization of Functional Mock-up Units**. 9th International Modelica Conference. Munich, 2012.
- [7] T. Schierz, M. Arnold, C. Clauss: **Co-simulation with Communication Step Size Control in an FMI Compatible Master Algorithm**. 9th International Modelica Conference. Munich, 2012.
- [8] S. Burhenne, M. Pazold, F. Antretter, F. Ohr, S. Herkel, J. Radon: **WUFI Plus Therm: Co-Simulation unter Verwendung von Modelica Modellen**. Presentation at the Symposium „Integrale Planung und Simulation in Bauphysik und Gebäudetechnik.“ Dresden, March 2012.

Annex

Members of the FMI Modelica Association Project:

Project Leader	Torsten Blochwitz (ITI GmbH Dresden, Germany)
Steering Committee	Atego, Daimler, Dassault Systèmes, IFP EN, ITI, LMS, Modelon, QTronic, SIMPACK
Advisory Board	Armines, DLR, Fraunhofer (IIS/EAS, First, SCAI), Open Modelica Consortium, TWT, University of Halle
Guests	Altair Engineering, Berkeley University, Bosch, ETAS, Siemens, Equa Simulation

The Steering Committee is open for additional members that actively support FMI. Requirements: Must have (a) participated at least at two FMI meetings in the last 24 months, (b) must either provided the FMI standard or part of it in a commercial or open source tool, and/or must actively use FMI in industrial projects, (c) the Steering Committee members agree with qualified majority.

The Advisory Committee is open for additional members that proofed to actively support FMI. Requirements: Must have (a) participated at least at two FMI meetings in the last 24 months, and (b) the Steering Committee members agree with qualified majority.

Contributors to the FMI 2.0 Specification:

The following persons participated at FMI 2.0 design meetings and contributed to the discussion (alphabetical list):

Martin Arnold, University Halle, Germany
 Johan Akesson, Modelon, Sweden
 Mongi Ben-Gaid, IFP, France
 Torsten Blochwitz, ITI GmbH Dresden, Germany
 Christoph Clauss, Fraunhofer IIS EAS, Germany
 Alex Eichberger, SIMPACK AG, Germany
 Hilding Elmqvist, Dassault Systèmes AB, Sweden
 Markus Friedrich, SIMPACK AG, Germany
 Peter Fritzon, PELAB, Sweden
 Andreas Junghanns, QTronic, Germany
 Petter Lindholm, Modelon, Sweden
 Kristin Majetta, Fraunhofer IIS EAS, Germany
 Sven Erik Mattsson, Dassault Systèmes AB, Sweden
 Jakob Mauss, QTronic, Germany
 Dietmar Neumerkel, Daimler AG, Germany
 Peter Nilsson, Dassault Systèmes AB, Sweden
 Hans Olsson, Dassault Systèmes AB, Sweden
 Martin Otter, DLR (RMC-SR), Germany
 Bernd Relovsky, Daimler AG, Germany
 Tom Schierz, University Halle, Germany
 Bernhard Thiele, DLR (RMC-SR), Germany
 Antoine Viel, LMS International, Belgium

The following people contributed with comments (alphabetical list):

Peter Aaronsson, MathCore, Sweden
 Bernhard Bachmann, University of Bielefeld, Germany
 Iakov Nakhimovski, Modelon, Sweden
 Andreas Pfeiffer, DLR (RMC-SR), Germany

Generation of Sparse Jacobians for the Function Mock-Up Interface 2.0

J. Åkesson^{a,c}, W. Braun^d, P. Lindholm^b, B. Bachmann^d
^aLund University, Department of Automatic Control, Lund, Sweden
^bLund University, Department of Mathematics, Lund, Sweden
^cModelon AB, Lund, Sweden
^dUniversity of Applied Sciences Bielefeld, Bielefeld, Germany

Abstract

Derivatives, or Jacobians, are commonly required by numerical algorithms. Access to accurate Jacobians often improves the performance and robustness of algorithms, and in addition, efficient implementation of Jacobian computations can reduce the over-all execution time. In this paper, we present methods for computing Jacobians in the context of the Functional Mock-up Interface (FMI), and Modelica. Two prototype implementations, in JModelica.org and OpenModelica are presented and compared in industrial as well as synthetic benchmarks.

Keywords: FMI; Analytic Jacobians; Automatic Differentiation; JModelica.org; OpenModelica;

1 Introduction

Algorithms for solving computational problems numerically often require access to derivatives, or approximations thereof. Examples include simulation algorithms, where implicit integration schemes use derivative information in Newton type algorithms, optimization algorithms, where derivatives are used to compute search directions, and steady-state solvers. The quality of the derivatives typically affects performance and robustness of such algorithms. Often, the execution time is strongly affected by the calculation time of Jacobians.

During the last two years, the Functional Mock-up Interface ¹ (FMI) standard has had a strong impact amongst software tools for modeling and simulation. The goal of the standard is to promote model reuse and tool interoperability by providing a tool and language independent exchange format for models in compiled or source code form. Following the introduction of

FMI 1.0 in January 2010, the next version of the standard, FMI 2.0, will support sparse Jacobians, in order to enable increased efficiency of algorithms supporting FMI. The target of this extension is to provide derivative information for two different use cases of Functional Mock-up Units (FMUs). The first use case is simulation of a single FMU. In this case, sparse Jacobians for the model equations enable increased efficiency of iterative integration algorithms. The second use case is the composition of multiple FMUs, potentially blended also by elements from a modeling language such as Modelica, where directional derivatives are useful in order to efficiently construct Jacobians for systems of equations spanning several FMUs.

In this paper, we describe methods for generating sparse Jacobians and directional derivatives to fulfill the corresponding requirements of FMI 2.0. The methods are described in the context of compilation of Modelica models into FMUs, although the employed techniques are generally applicable to other model description formats. Two prototype implementations, one in OpenModelica² and one in JModelica.org³ are presented. The implementations of sparse Jacobians in the respective tools are compared based on industrial benchmark models.

The paper is organized as follows. In Section 2, material on FMI, Jacobians and differentiation techniques are provided. Section 3 describes two different implementations of sparse Jacobians in JModelica.org and OpenModelica respectively. Benchmark results are provided in Section 4, and the paper ends with a summary and conclusions in Section 5.

¹<https://fmi-standard.org/>

²<http://www.openmodelica.org>

³<http://www.jmodelica.org>

2 Background

2.1 The Functional Mock-up Interface

FMI emerged as a new standard resulting from the ITEA2 project MODELISAR, in 2010. The standard is a response to the industrial need to connect different environments for modeling, simulation and control system design. Commonly, different tools are used for different applications, whereas simulation analysis at the system integration level requires tools to be connected. FMI provides the means to perform such integrated simulation analysis.

FMI specifies an XML format for model interface information and a C API for model execution. The XML format, specified by an XML schema, contains information about model variables, including names, units and types, as well as model meta data. The C API, on the other hand, contains C functions for data management, e.g., setting and retrieving parameter values, and evaluation of the model equations. The implementation of the C API may be provided in source code format, or more commonly as a compiled dynamically linked library.

FMI comes in two different flavors: FMI for Model Exchange (FMI-ME) [2] and FMI for Co-Simulation (FMI-CS) [3]. FMI-ME exposes a hybrid Ordinary Differential Equation (ODE), which may be integrated stand-alone or which may be incorporated in a composite dynamic model in a simulation environment. The FMI-ME C API exposes functions for computation of the derivatives of the ODE, and accordingly, in FMI-ME the integration algorithm is provided by the importing application. FMI-CS, on the other hand, specifies that the integration algorithm is included in the FMU, and the FMU-CS C API provides functions for integrating the dynamics of the contained ODE for a specified period of time.

The FMI standard is supported by several modeling and simulation tools, including Dymola, SimulationX, JModelica.org and OpenModelica. Also, there are FMI interfaces to MATLAB, National Instruments Veristand and several additional tools.

FMI 2.0 is a unification of the Model Exchange and Co-simulation standards and contains several improvements. One of those are the sparse Jacobians, which are also topic of this paper. The sparse Jacobian interface in FMI 2.0 consists of three different parts:

- A C API function for evaluation of directional derivatives of the model equations.
- A C API function for evaluation of sparse Jaco-

bian matrices corresponding to the ODE representation of an FMU.

- A section in the XML document contained in an FMU providing the incidence pattern for the Jacobian matrices.

In this paper, algorithms for generating this functionality are discussed.

2.2 Causalization of DAEs

In the first step of the compilation process in a Modelica tool chain, a compiler front-end transforms Modelica source code into a flat representation, consisting essentially of lists of variables, functions, equations and algorithms. Based on this model representation, symbolic operations such as alias elimination and index reduction are applied, in order to reduce the size of the model and to ensure that the resulting Differential Algebraic Equation (DAE) is of index 1. In this section, we outline the following steps that are of particular relevance for the generation of Jacobians. In particular, the causalization procedure, i.e., transformation of an index-1 DAE into an equivalent ODE, as required by the FMI standard, is discussed.

FMI specifies Jacobians and directional derivatives with respect to the continuous model equations. Therefore, without lack of generality, and for clarity of the presentation, only the continuous part of the DAE is considered in the following.

We consider index-1 DAEs in form of

$$\begin{aligned} F(\dot{x}(t), x(t), u(t), w(t)) &= 0, & t \in [t_0, t_f] \\ x(0) &= x_0 \end{aligned} \quad (1)$$

where $\dot{x}(t) \in R^{n_x}$ are the state derivatives, $x(t) \in R^{n_x}$ is the state, $u(t) \in R^{n_u}$ are the inputs and $w(t) \in R^{n_w}$ are the vector of algebraic variables. The initial conditions of DAE state is given by x_0 . Introducing $z = (\dot{x} \ w)$, denoting the unknowns of the DAE, and $v = (x \ u)$, denoting the known variables, the DAE written

$$F(z, v) = 0 \quad (2)$$

The conceptual idea of DAE causalization commonly used in Modelica tools is then to compute the inverse relationship of F

$$z = G(v), \quad (3)$$

and the ODE may then be written

$$\begin{aligned} \dot{x} &= f(x, u) \\ y &= h(x, u) \end{aligned} \quad (4)$$

where y are the outputs of the system. Note that the algebraic variables are considered to be internal to the ODE in this representation. In general, there is no closed expression for the functions f and g , but rather, iterative techniques, e.g., Newton's method, is employed to solve algebraic loops for z .

Modelica models are typically of large scale but sparse in the sense that each model equation contains references only to a small number of equations. In order to exploit this structure, graph algorithms can be employed. Two commonly used algorithms that are used for this purpose are *matching algorithms*, e.g., the Hopcroft Karp algorithm, and Tarjan's algorithms for computing *strong components*, [4]. The result of Tarjan's algorithm is then used to permute the variables and equations of the DAE into Block Lower Triangular (BLT) form.

Let us consider a DAE with five equations and five unknowns, i.e., $F \in R^5$ and $z \in R^5$, where the DAE equations are given by

$$\begin{aligned} F_1(z_1, z_5, v) &= 0 \\ F_2(z_3, v) &= 0 \\ F_3(z_1, z_2, z_3, z_4, v) &= 0 \\ F_4(z_1, z_3, z_5, v) &= 0 \\ F_5(z_2, z_5, v) &= 0 \end{aligned} \quad (5)$$

Note that the variables $v = [x, u]$ are known and need not be considered in the following analysis. The dependence of the z -variables can be shown in the following incidence matrix,

	z_1	z_2	z_3	z_4	z_5
F_1	*	0	0	0	*
F_2	0	0	*	0	0
F_3	*	*	*	*	0
F_4	*	0	*	0	*
F_5	0	*	0	0	*

(6)

A * in the incidence matrix at row i and column j denotes that the residual function F_i contains a reference to the variable z_j . Application of the BLT procedure, now yields the following DAE system

	z_3	z_1	z_5	z_2	z_4
F_2	1	0	0	0	0
F_4	1	1	1	0	0
F_1	0	1	1	0	0
F_5	0	0	1	1	0
F_3	1	1	0	1	1

(7)

The implicit DAE system (5) is now given by a sequence of assignment statements and implicit systems

of equations

$$\begin{aligned} \bar{z}_1 &:= g_1(v) \\ \bar{F}_2(\bar{z}_1, \bar{z}_2, v) &= 0 \\ \bar{F}_3(\bar{z}_2, \bar{z}_3, v) &= 0 \\ \bar{z}_4 &:= g_4(\bar{z}_1, \bar{z}_2, \bar{z}_3, v) \end{aligned} \quad (8)$$

where $\bar{z}_1 = z_3$, $\bar{z}_2 = (z_1 \ z_5)^T$, $\bar{z}_3 = z_2$, $\bar{z}_4 = z_4$. The functions g_1 and g_2 corresponds to explicit solutions of the corresponding DAE equations, whereas $\bar{F}_2 = (F_4 \ F_1)^T$ and $\bar{F}_3 = F_5$ corresponds to implicit (systems of) equations that require iteration. It is typical for Modelica models to contain only a small number of implicit systems of equations and a large number of trivial, e.g., linear equations that may be solved symbolically.

For a general DAE, the BLT procedure results in a sequence of scalar and non-scalar equation blocks on the form

$$\begin{aligned} \bar{F}_1(\bar{z}_1, v) &= 0 \\ &\vdots \\ \bar{F}_i(\bar{z}_1, \dots, z_i, v) &= 0 \\ &\vdots \\ \bar{F}_b(\bar{z}_1, \dots, z_b, v) &= 0 \end{aligned} \quad (9)$$

where the unknown of each residual \bar{F}_i is \bar{z}_i . Further, some of the residual functions may be solved explicitly by symbolic manipulation and the remaining blocks needs the to be solved by iterative methods.

Computation of the sequence of solved and non-solved blocks (9), given values of the known variables in v then produces the corresponding state derivative and algebraic vectors contained in z . Accordingly, the DAE has been causalized in to an ODE on the form (4).

2.3 Computation of Jacobians

The Jacobian of a vector valued function $f(x) \in R^m$, $x \in R^n$ is given by

$$\frac{\partial f}{\partial x} = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \dots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \dots & \frac{\partial f_m}{\partial x_n} \end{pmatrix} \quad (10)$$

A useful tool when computing Jacobians is directional derivatives. The directional derivative of a vector valued function $f(x)$ is defined by

$$df = \frac{\partial f}{\partial x} \cdot dx, \quad (11)$$

where $dx \in R^n$ represents the direction in which the directional derivative, denoted $df \in R^m$, is evaluated. dx is also referred to as a seed vector.

In the following, directional derivatives will be used extensively to construct Jacobians. A straight forward, although naive, approach to construct a Jacobian from directional derivative evaluations is as follows. Using the identity matrix I of dimension n , and the unit vectors $e_1 \dots e_n$ we have that

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial x} I = \frac{\partial f}{\partial x} \begin{pmatrix} e_1 & \dots & e_n \end{pmatrix} = \begin{pmatrix} \frac{\partial f}{\partial x} \cdot e_1 & \dots & \frac{\partial f}{\partial x} \cdot e_n \end{pmatrix}. \quad (12)$$

Using this relation, a Jacobian with n columns may be constructed from n evaluations of directional derivatives. In Section 2.7, an overview of methods to explore sparsity to improve efficiency in this respect will be given.

There are three widely used methods for computing Jacobians, namely finite difference methods, symbolic differentiation and automatic (or algorithmic) differentiation.

2.4 Finite Difference Approximation

In the finite difference method, a numerical approximation of the directional derivative of a vector valued function f is calculated using the formula

$$\frac{\partial f(x)}{\partial x} \cdot e_i = \frac{f(x + e_i h) - f(x)}{h}. \quad (13)$$

where h is the increment. On one hand, even if the increment is chosen optimal in nature of that method is an accuracy error ε , which is the sum of $\varepsilon_t + \varepsilon_r$ where ε_t is the truncation error and ε_r the round-off error. The truncation error $\varepsilon_t |h^i f(x)|$ is the result of the Taylor-series truncation. The round-off error $\varepsilon_r \varepsilon_f |f(x)/h|$ where ε_f is the fractional accuracy $\varepsilon_f \geq \varepsilon_m$ depends on machine accuracy ε_m . On the other hand, it is easy to implement and also almost applicable.

2.5 Symbolic Differentiation

In general the “calculus” of symbolic derivatives is done by difference quotients. where the derivative of a function is the limit

$$\frac{\partial f}{\partial x} = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h} \quad (14)$$

difference quotients. This is also the way the basic differentiation rules are found. From a practical view the

“calculus” of the symbolic derivatives is done by applying basic differentiation rules and table of derivatives for common functions on the expressions to find the formulas for the derivatives. Since a Modelica model results during the compilation in symbolic expressions which are manipulated to simplify the original system. So it is quite typical for a Modelica Tool to use symbolical methods also for the differentiation. Finding the symbolic formula may take time, space and a symbolic kernel for simplifications, but once determined it’s fast to evaluate them [7]. A further drawback is that symbolic differentiation is not applicable on algorithms (with for-loops and branches).

2.6 Automatic Differentiation

Automatic Differentiation (AD) is a method for computing derivatives with machine precision, which is applicable to expressions as well as algorithmic functions [1]. The key idea in AD techniques is to propagate derivative information through a sequence of atomic operations, which is represented by an *expression graph*. Computation of a sequence of AD operations results in the evaluation of a directional derivative with respect to a given seed vector.

There are two different modes of operation of AD—forward and reverse. The forward mode AD is conceptually simple, and is based on forward propagation of values and derivatives through an expression graph. The result of a forward AD sweep is a vector corresponding to the Jacobian multiplied by the seed vector. Commonly, Jacobian matrices are constructed from a number of forward AD evaluations.

The reverse AD technique is more involved than the forward mode, and includes a forward and a backward evaluation sweep over the expression graph, and the result is a vector corresponding to the seed vector multiplied by the Jacobians. This mode of operation is particularly useful in the case of scalar functions that depends on many independent variables—in this case, reverse AD is referred to the cheap gradient computation. Reverse AD is also commonly used to construct higher-order derivatives, e.g., Hessian matrices in optimization applications.

Implementation of AD tools comes two different flavors: Operator Overloading (OO) and Source Code Transformation (SCT). In OO tools, the expression graph is represented by data structures that are repeatedly traversed during forward and reverse mode evaluations. This strategy has been popularized by tools

such as CppAD⁴ and ADOL-C⁵ which both enable AD to be applied to C code with minor modifications. Tools in this category are typically based on operator overloading, e.g., in C++, to construct a data structure referred to as a *tape*, which is then used as a basis for derivative computations. Tools based on the SCT approach, on the other hand, generate code that, when executed, compute derivatives. The ADIFOR⁶ package falls into this category.

In this paper, forward mode AD using the SCT technique will be used. The remainder of this section will therefore focus on explaining this methods.

A key to understanding forward AD, is the observation that expressions can be evaluated, and differentiated, by considering a sequence of atomic operations. The elementary arithmetic operations can be differentiated by applying the derivation rules

$$\begin{aligned}\frac{d}{dx}(u(x) \pm v(x)) &= \frac{du}{dx} \pm \frac{dv}{dx} \\ \frac{d}{dx}(u(x)v(x)) &= u(x)\frac{dv}{dx} + v(x)\frac{du}{dx} \\ \frac{d}{dx}\left(\frac{u}{v}\right) &= \frac{v(x)\frac{du}{dx} - u(x)\frac{dv}{dx}}{v(x)^2}\end{aligned}$$

In addition, the chain rule

$$\frac{d}{dx}\phi(u(x)) = \frac{d\phi}{du} \frac{du}{dx}$$

applies to the elementary arithmetic functions, such as sin, cos etc.

In the following example, we illustrate how these building blocks are used to apply the forward AD technique. We consider the function

$$f(x_1, x_2) = x_1 \cdot x_2 + \sin(x_1), \quad (15)$$

for which we would like to compute the directional derivative according to relation (11). Assuming the seed vector $dx = (1 \ 0)^T$, it follows that

$$df = \frac{\partial f}{\partial x} dx = \begin{pmatrix} \frac{\partial f}{\partial x_1} & \frac{\partial f}{\partial x_2} \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \frac{\partial f}{\partial x_1}. \quad (16)$$

Using the seed vector in (16), $f(x)$ will be differentiated with respect to x_1 .

The expression graph corresponding to the function in (15) is shown in Figure 1.

In the figure, the leaves represent the independent variables and the root node represents the function itself.

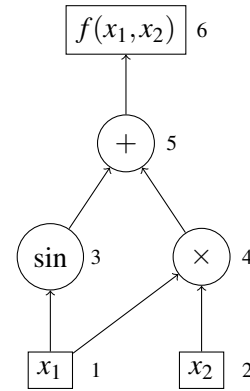


Figure 1: Expression graph of the function (15)

A forward AD sweep is performed as follows. The computation sequence starts at the independent variables. Intermediate variables, v_i 's, are introduced to hold the value of each node, and in addition, variables for the derivative values of each node, d_i , are introduced. The expression of a particular variable v_i is given by the corresponding node type, i.e., arithmetic operation, and the derivative value, d_i , is given by differentiation of the same operation. Application of this procedure to the function (15) gives the following sequence of operations.

$$\begin{aligned}v_1 &:= x_1 \\ d_1 &:= dx_1 \\ v_2 &:= x_2 \\ d_2 &:= dx_2 \\ v_3 &:= \sin(v_1) \\ d_3 &:= d_1 \cdot \cos(v_1) \\ v_4 &:= v_1 \cdot v_2 \\ d_4 &:= d_1 \cdot v_2 + v_1 \cdot d_2 \\ v_5 &:= v_3 + v_4 \\ d_5 &:= d_3 + d_4 \\ v_6 &:= v_5 \\ d_6 &:= d_5\end{aligned}$$

The variable v_6 now holds the value of the function itself and d_6 holds the value of the directional derivative. Note that the evaluation is done for particular values of the independent variables, in this case x_1 and x_2 , and seed values, dx_1 and dx_2 . Note that auxiliary variables v_1, v_2, d_1 and d_2 are introduced here for clarity.

2.7 Exploiting Sparsity

Modelica models, also after the causalization procedure described above, are often sparse, i.e., each equa-

⁴<http://www.coin-or.org/CppAD/>

⁵<http://www.coin-or.org/projects/ADOL-C.xml>

⁶<http://www.mcs.anl.gov/research/projects/adifor/>

tion of a model depends only on a fraction of the total number of variables. Exploiting sparsity of Modelica models can be done in two different contexts. Firstly, the efficiency of computation of Jacobian matrices based on directional derivative evaluations can be much improved by considering sparsity. This strategy is called *compression* and will be described briefly in this section. Secondly, a simulation environment importing an FMU providing sparse Jacobians may utilize this information to improve the performance of numerical algorithms. A typical example of such algorithms are sparse linear solvers, e.g., UMFPACK⁷, CSparse⁸ and PARDISO⁹. This usage is, however, not related to the procedures required to generate Jacobians, and it is therefore beyond the scope of this paper.

As noted above, a naive method for evaluation directional derivatives to generate Jacobian matrices is to simply make one such evaluation for each column of the Jacobian, with seed vectors corresponding to the unit vectors of appropriate dimension. If the Jacobian is sparse, however, the number of evaluations can be drastically reduced, by observing that several columns can be computed in a single directional derivative evaluation if the sparsity patterns of these columns do not overlap. As an example, consider the incidence matrix (6). Here, we note that columns four and five does not contain overlapping entries, and they can therefore be computed by one single directional derivative evaluation with the seed vector chosen as the sum of the corresponding unit vectors. Note also that this strategy is applicable to all three differentiation methods described above: finite differences, AD and symbolic differentiation.

While this strategy is simple to implement, computing a column grouping of minimal size is well known to be an NP-hard problem—this problem corresponds precisely to the graph coloring problem [5, 6]. There are, however, efficient algorithms capable of computing practically useful approximations of the optimal solutions. Specific algorithms will be discussed in Section 3.

3 Computation of Jacobians for Modelica Models

In Section 2.2, it was shown how a DAE is transformed into an ODE by means of the BLT transformation. In this section, methods for computing the Jacobians of the resulting ODE (4) are presented. We consider

$$\frac{\partial z}{\partial v} = \begin{pmatrix} \frac{\partial \dot{x}}{\partial v} \\ \frac{\partial y}{\partial v} \end{pmatrix} = \begin{pmatrix} \frac{\partial \dot{x}}{\partial x} & \frac{\partial \dot{x}}{\partial u} \\ \frac{\partial w}{\partial x} & \frac{\partial w}{\partial u} \end{pmatrix} = \begin{pmatrix} A & B \\ C & D \end{pmatrix} \quad (17)$$

In this section, we present two methods for computing the matrices A , B , C and D by means of directional derivatives. One of the methods, which is implemented in JModelica.org, relies on a forward AD implementation in an SCT setting, whereas the other method, which is implemented in OpenModelica, relies on symbolic differentiation and symbolic expression simplification. In addition, an algorithm for computing the sparsity pattern of the Jacobian matrices, which is common for both methods, is presented.

The key idea in this section is the following. Differentiating the DAE (2) yields the relation

$$\frac{\partial F}{\partial z} dz + \frac{\partial F}{\partial v} dv = 0, \quad (18)$$

where dv is the input seed vector and dz works as the directional derivative of the relation (3) with respect to the direction dv . By solving the system of equations (18) for a particular seed dv , the directional derivative of the DAE is obtained. It is important to note that the system of equations to be solved is linear in the unknowns, dz , and thus does not require iteration.

The Jacobian matrices are then constructed from repeated evaluation of directional derivatives. In addition, coloring algorithms and compression is used to reduce the number of directional derivative evaluations in both implementations.

Evaluation of Jacobians based on the compression of the columns requires access to sparsity pattern, as stated in Section 2.7. The determination of the sparsity pattern for a Modelica model could be done by means of graph theory. Since the non-zero values in a Jacobian expresses which output variable has a connection to which input variable. Thus the determination problem could be formulated as a st-connectivity problem in a directed graph, where input variables are the sources and the output variables are the sinks. The st-connectivity is a decision problem that asks if the vertex t is reachable from the vertex s , particular which output variable is connected to which input variable. Specific algorithms for this purpose will be discussed below.

⁷<http://www.cise.ufl.edu/research/sparse/umfpack/>

⁸http://people.sc.fsu.edu/~jburkardt/c_src/csparse/csparse.html

⁹<http://www.pardiso-project.org/>

3.1 Implementation of Directional Derivatives in JModelica.org

The performance of the approach outlined above can be improved significantly by exploiting the BLT structure described in Section 2.2. In particular, forward AD may be applied directly to the sequence of computations given in (9). In the implementation in JModelica.org, C code corresponding to a forward AD sweep over the sequence of BLT blocks is generated. The symbolic expression graphs in the compiler is a basis for the code generation. As noted in Section 2.2, there are two kinds of blocks produced by the BLT transformation, i.e. solved equation blocks and non-solved equation blocks requiring iterative numerical solution. Below, we explain how directional derivatives are propagated in these two cases.

3.1.1 Propagation of Directional Derivatives in Equation Blocks

For blocks corresponding to solved equation blocks of the form

$$\bar{z}_i := g_i(\bar{z}_1, \dots, \bar{z}_{i-1}, v) \quad (19)$$

it is straight forward to apply the forward AD approach. In this case, AD code is simply generated basing on the expression graph for g_i , in order to produce the directional derivative

$$d\bar{z}_i = \frac{\partial g_i}{\partial \bar{z}_1} d\bar{z}_1 + \dots + \frac{\partial g_i}{\partial \bar{z}_{i-1}} d\bar{z}_{i-1} + \frac{\partial g_i}{\partial v} dv. \quad (20)$$

Note that the input seed dv and the directional derivatives for previous blocks, $\bar{z}_1, \dots, \bar{z}_{i-1}$ are known at this point in the computation sequence. Commonly, the expression g_i does not depend on all previous vectors of unknowns, $\bar{z}_1, \dots, \bar{z}_{i-1}$, a property which is exploited in the implementation.

For a block corresponding to a system of equations, the block residual is given by

$$\bar{F}_i(\bar{z}_1, \dots, z_i, v) = 0. \quad (21)$$

In order to compute the directional derivative, $d\bar{z}_i$, for such a block, the residual equation is differentiated to yield

$$\frac{\partial \bar{F}_i}{\partial \bar{z}_1} d\bar{z}_1 + \dots + \frac{\partial \bar{F}_i}{\partial \bar{z}_i} d\bar{z}_i + \frac{\partial \bar{F}_i}{\partial v} dv = 0 \quad (22)$$

which in turn gives the linear system

$$\frac{\partial \bar{F}_i}{\partial \bar{z}_i} d\bar{z}_i = - \sum_{k=1}^{i-1} \frac{\partial \bar{F}_i}{\partial \bar{z}_k} d\bar{z}_k - \frac{\partial \bar{F}_i}{\partial v} dv \quad (23)$$

to be solved for $d\bar{z}_i$. All Jacobians in this relation are generated to C code using forward AD. Note that the system Jacobian of the linear system (23) is provided also to the Newton solver that computes the solution of the system of equations (21). Therefore, this code is reused in the computation of the directional derivative of the block.

3.1.2 Computation of Sparsity Patterns

Computation of sparsity patterns for the Jacobian matrices A , B , C and D is a non-trivial problem, because of the sequence of operations required to compute the state derivatives x and the algebraic variables w . In comparison, computation of the Jacobian matrix of a DAE system (2) is straightforward and can be done by simply collecting references to unknown variables in each residual equation. As noted above, the problem of computing sparsity patterns for the ODE Jacobian is a connectivity problem, where the dependencies of the dependent variables z of the independent variables contained in v need to be computed.

The BLT form of the DAE offers means to compute the required sparsity patterns for the ODE Jacobians. While the general form of a block in the BLT sequence is

$$\bar{F}_i(\bar{z}_1, \dots, z_i, v) = 0, \quad (24)$$

particular blocks typically do not depend on all variables in z_1, \dots, z_i and in v . In order to reflect this situation, we introduce the notation

$$\tilde{F}_i(\tilde{z}_i, z_i, \tilde{v}_i) = 0, \quad (25)$$

where \tilde{z}_i contains the variables in the z vector upon which the equation block residual \tilde{F}_i depends. \tilde{v}_i is defined correspondingly. As a first approximation, which will be relaxed in the following, we assume that all variables solved for in the block i , i.e., z_i , depends on all variables in \tilde{z}_i and in \tilde{v}_i . Clearly, this relationship defines the direct dependency of z_i on \tilde{v}_i . Now, the dependency of z_i on the variables contained in $\tilde{v}_1, \dots, \tilde{v}_{i-1}$ is given implicitly by \tilde{z}_i . The connectivity graph of the BLT form reveals these dependencies. Edges in this graph corresponds to non-zero entries in the lower left part of the transformed incidence matrix, below the block diagonal. In the connectivity graph, dependency information is propagated top-down in the sequence of blocks. For each block, the complete set of variables in v upon which the block depends is collected from the predecessors in the dependency graph.

For a block consisting of a system of equations, the assumption that all variables solved for in the block,

z_i , depends on all variables in \tilde{z}_i may lead to an over-estimation of the sparsity pattern. Specifically, since the sparsity pattern of the inverse of a sparse matrix may also be sparse, the computation may result in non-zero entries which are in fact structural zeros. In order to take this into account, the sparsity pattern of the inverse of the corresponding block Jacobian may be computed, [8]. The result of this analysis is then taken into account when variable dependencies are computed. Note that this analysis remains to be implemented in JModelica.org

3.2 Implementation of Directional Derivatives in OpenModelica

The directional derivatives in OpenModelica are generated basically by setup a new symbolic equation system inside the OMC with the differentiated equations. This system contains the desired partial derivatives $d\tilde{z}$ as unknowns, the seed vector $d\tilde{v}$ and all other variables from the original system are considered as known. The resulting equation system is the desired one as in equation (18).

This approach differs from the previously published procedure (see [10]), in a way that now each equation is derived only once. This leads to linearity in the compilation time and in the generated code size.

All methods mentioned in section 2 are used for the differentiation of the original system. Equations are differentiated symbolically, algorithm sections and Modelica functions without an derivative annotation are differentiated by the forward AD approach and external functions, where nothing else is possible, are differentiated numerically.

The generated equation system is then optimized like the original system. In detail it is transformed to an explicit form with the BLT machinery of OpenModelica, further expression-based simplification are done and some common sub-expressions are filtered. The resulting equation system is then written to the C-Code.

For the purpose of generating the four matrices in (17) for each matrix one new equation system is generated with the corresponding variables. Note therefore the original system is filtered for the necessary equations.

The exploration of the sparsity pattern for a fast evaluation of the compressed Jacobians is applied on the generated directional derivatives. A detailed description of the algorithms used for that task in OpenModelica can be found in [9].

3.3 Comparison of Implementations

The implementations in OpenModelica and in JModelica.org share common characteristics, but there are also differences. Both algorithms are based on generation of C code that evaluates directional derivatives, which in turn are used to compute Jacobians. Also, both algorithms rely on compression for reducing the number of directional derivative evaluations. The computation of sparsity patterns for the ODE Jacobians also proceeds in the same manner.

The main difference between the implementations is rather the way in which the directional derivatives are generated. In the JModelica.org implementation, the same BLT structure as for the underlying ODE is used. Code generation is done by traversing the BLT structure in a separate code generation pass and forward AD code is then generated for solved equations and systems of equations, as described in 3.1. In the OpenModelica implementation, on the other hand, a new data structure containing all model equations in symbolically differentiated form is first constructed. The symbolic kernel of the compiler is then invoked to simplify the differentiated equations, and a new BLT structure is computed prior to code generation.

Both approaches have advantages and disadvantages. In the JModelica.org implementation, no new data structures are created, which reduces memory consumption. Also, since the same BLT structure as for the underlying ODE is used, Jacobians for systems of equations corresponding to algebraic loops are generated. These, in turn are useful also in case of applying iterative techniques to solve algebraic loops. The main advantage of the OpenModelica implementation is that symbolic simplifications done by the compiler can yield simpler code that is faster to execute. Also, since a new BLT computation is done, properties of the new, differentiated system of equations may be explored in order to further speed up Jacobian computations.

4 Benchmarks

Three different aspects are considered in the benchmarks presented in this section, namely, *i*) model compilation time, *ii*) generated code size, and *iii*) Jacobian evaluation time. In the case of model compilation time, both the time spent in the respective Modelica compilers, OpenModelica and JModelica.org, and the time spent in the C compiler, gcc in both cases, when compiling the generated code is measured. This

measure seems to be the most interesting for the user, since both phases are included in the model compilation time from a user's perspective. As for the size of the generated code, only the size of the code that is generated by the Modelica compilers is measured, i.e., no code originating from run-time systems or similar is included. Finally, the time for 1000 Jacobian evaluations is measured and the mean evaluation times are reported. In all benchmarks, the system Jacobian, i.e., the Jacobian of the derivatives with respect to the states, is evaluated.

It is worth noting that the benchmarks in this section does not only reflect the particular details of the respective Jacobian evaluation strategies. In particular, the measurements are biased by other code optimization strategies in the compilers, including alias elimination, symbolic processing, tearing, and the efficiency of non-linear solvers used to solve algebraic loops. In addition, the compilation time measurements, the optimization and debugging flags supplied to the respective C compilers influence the result.

All measurements in this paper are performed on a 64-bits architecture computer having one Intel Q9550@2.83GHz CPU and 16 GB of RAM. It runs Ubuntu 12.04 Linux, kernel 3.2.0-25.

4.1 Combined Cycle Power Plant

The first benchmark is a model of a combined cycle power plant model, see Figure 2. The model contains equation-based implementations of the thermodynamic functions for water and steam, which in turn are used in the components corresponding to pipes and and the boiler. The model also contains components for the economizer, the super heater, as well as the gas and steam turbines. The model has 10 states and 131 equations. For additional details on the model, see [11].

The benchmark results are shown in Table 1. As can be seen, the model compilation times and the file sizes are similar. Both implementations obtained six colors for the Jacobian, i.e., 6 directional derivative evaluations were required to compute the Jacobian. The Jacobian evaluation time does, however, differ in a way that the OpenModelica implementation performs faster.

4.2 Synthetic Benchmarks

In order to analyze the scalability properties of the respective implementations, synthetic benchmark models were automatically generated. The underlying as-

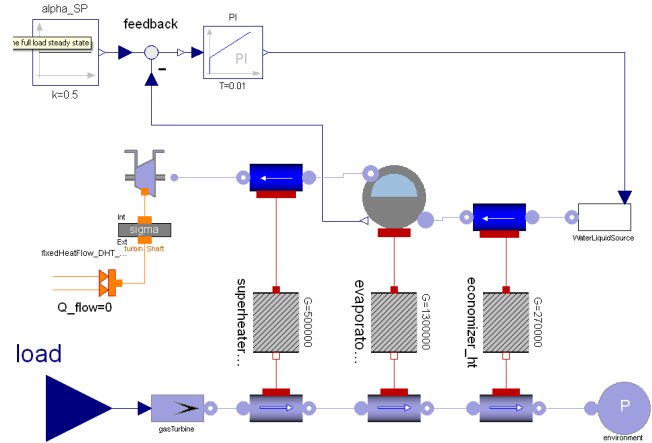


Figure 2: Modelica component diagram for a combined cycle power plant.

Table 1: Benchmark results for combined cycle power plant.

Tool	Generation [s]		Code size [kB]		Jac eval time[ms]
	No Jac	Jac	No Jac	Jac	
OM	2.98	3.87	519	711	0.018
JM	3.64	5.92	266	456	0.090

sumption of the synthetic models is that a single Modelica equation contains references to fixed maximum number of variables, a number which does not increase with model size. This assumption is realistic, given that Modelica models are typically constructed from a large number of simple component models, where the equations in each component are local in the sense that they refer mainly variables in the same, or neighboring, components. Another important feature of Modelica models are algebraic loops, or implicit systems of equations, which require iterative solution techniques. Therefore, the synthetic benchmark models contain implicit systems of equations, composed from linear and non-linear terms, in the form of *sin* functions, terms.

Three suits of benchmark models were constructed, using different assumptions on the number of variable references in a single equation. This aspect was quantified by the sizes of the implicit systems of equations, where sizes of two, four and eight, respectively, were used to generate the benchmark models. Within each suite of benchmark models, four different models of increasing size were constructed, essentially by doubling the number of variables while keeping the size of all the implicit equation systems constant. For detailed statistics and structural analysis of the models

Table 2: Statistics and structural analysis of the synthetic models. #N denotes the number of variables, #N-z. denotes the number of reported non-zero elements and #Col. denotes the number of colors resulting from the coloring algorithms. #N-z. and #Col. are equal in both implementations.

	#N	#States	#Alg. loops	#N-z.	#Col.
1-1	22	4	9	7	2
1-2	42	8	17	21	4
1-3	82	16	33	47	4
1-4	162	32	65	104	4
2-1	40	4	9	7	2
2-2	76	8	17	21	4
2-3	148	16	33	53	4
2-4	292	32	65	117	4
3-1	76	4	9	7	2
3-2	144	8	17	21	4
3-3	280	16	33	53	4
3-4	552	32	65	117	4

consider table 2. Note that for both implementations, the number of non-zero elements and the number of colors produced by the respective coloring algorithms are equal.

Table 3: Benchmarks of synthetic models for OpenModelica

	Generation [s]		Code size [kB]		Jac. eval.
	No Jac.	Jac.	No Jac.	Jac.	time [ms]
1-1	0.57	1.3	41	121	0.008
1-2	0.87	2.0	72	225	0.033
1-3	1.51	3.7	134	435	0.068
1-4	2.82	7.4	260	860	0.142
2-1	0.88	2.1	64	208	0.017
2-2	1.51	4.1	114	393	0.067
2-3	2.75	7.5	218	781	0.144
2-4	5.36	15.5	429	1569	0.308
3-1	2.22	6.6	117	457	0.048
3-2	4.20	13.7	219	889	0.198
3-3	8.45	27.7	432	1789	0.421
3-4	17.02	56.9	857	3583	0.873

The results in terms of model compilation time, generated code size, and Jacobian evaluation time for the different models are shown in Tables 3 and 4, for the OpenModelica and the JModelica.org implementations respectively. Figures 3, 4, and 5 depict the corresponding results graphically. Each curve corresponds to one benchmark suite. As can be seen from the tables, all three measures exhibit essentially lin-

Table 4: Benchmarks of synthetic models for JModelica.org

	Generation [s]		Code size [kB]		Jac. eval.
	No Jac.	Jac.	No Jac.	Jac.	time [ms]
1-1	1.02	1.88	36	138	0.037
1-2	1.24	3.16	54	247	0.089
1-3	1.78	5.71	93	484	0.163
1-4	3.16	10.71	171	957	0.316
2-1	1.37	4.39	61	388	0.104
2-2	2.04	8.17	102	737	0.334
2-3	3.44	15.35	187	1435	0.673
2-4	6.42	31.52	360	2843	1.269
3-1	3.05	15.65	146	1371	0.558
3-2	5.28	30.33	264	2581	2.078
3-3	9.93	63.49	511	5146	4.027
3-4	19.66	136.05	1009	10315	8.827

ear complexity for a fixed size of the algebraic loops. This result is the key to scalability of the methods. The smallest model in each benchmark suite deviates from the linear trend for Jacobian evaluation time, which is due to the fact that fewer colors are needed in these cases.

While model compilation time and generated code size without Jacobians are similar in all cases for OpenModelica and JModelica.org, the corresponding numbers with Jacobians differ. The difference in code size is due to the fact that JModelica.org relies on generation of forward AD code, without simplifications, which results in verbose code. Also, inherent in the forward AD strategy is that both the model equations in their original form and the directional derivatives are evaluated simultaneously. In comparison, the OpenModelica implementation differentiates the equations symbolically and then applies symbolic simplification. In this case, the resulting expressions that are generated are simpler, and also, no additional code is generated for the original model equations.

In terms of execution speed, the OpenModelica implementation performs faster. The main reason for this is that the application of forward AD in the JModelica.org implementation results in more verbose code, and also the model equations, along with the directional derivatives, are evaluated.

It is worth noting that either the effect of different versions of LAPACK/BLAS, used to solve linear systems in both implementations, nor the the influence of different compiler optimization and debugging flags have been considered in the benchmarks. Rather, the performance experienced by users has been reported.

Both implementations may be further optimized in these respects in order to improve compilation and execution times. Therefore, the reported benchmarks do not solely measure the efficiency of the respective methods described in the paper, but are rather biased with the details of the particular implementations.

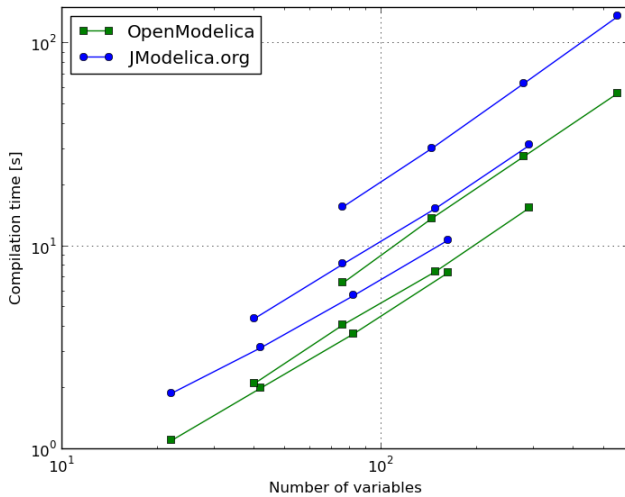


Figure 3: Model compilation time with Jacobians.

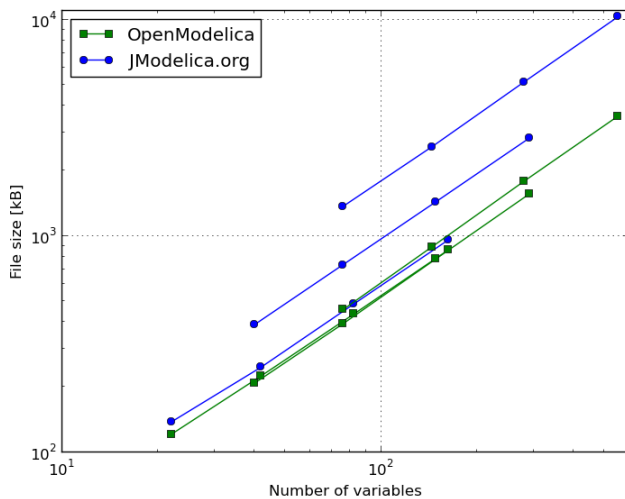


Figure 4: Generated code size with Jacobian.

5 Conclusions

In this paper, the generation of Jacobians for ODEs originating from DAEs, in particular Modelica models, has been discussed. The algorithmic machinery employed consists of known methods and algorithms, such as numerical, symbolic, and automatic differen-

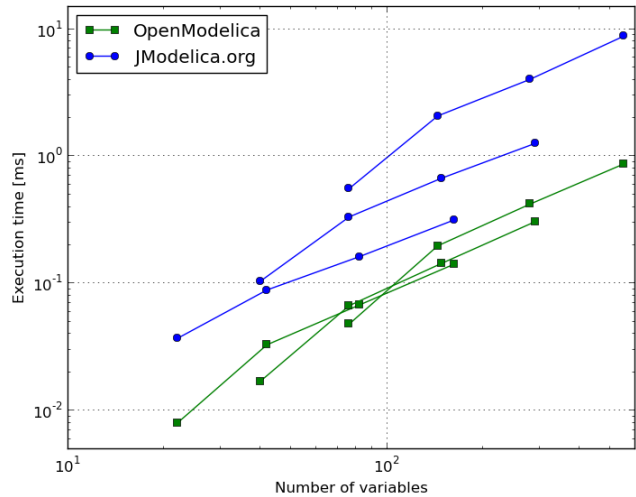


Figure 5: Execution time for one Jacobian evaluation

tiation, as well as graph theoretic methods such as the BLT transformation. Two methods, sharing similarities as well as differences have been presented. One of the methods is a straight forward application of forward automatic differentiation and generation of C code, which results in functions for the evaluation of directional derivatives, which in turn are used to compute Jacobians. The other method relies mainly on symbolic differentiation and makes use of symbolic simplification algorithms in a Modelica compiler to generate directional derivative functions. Both methods provide sparsity patterns for the ODE Jacobians, and they both make efficient use of sparsity in order to reduce the number of directional derivative evaluations, a technique referred to as compression.

The two approaches are implemented in JModelica.org and OpenModelica, respectively, and compared in an industrial benchmark as well as in several synthetic benchmarks. Both implementations show linear growth in key measures such as model compilation time, generated code size and execution time, under realistic assumptions on model structure. In terms of execution speed, the method relying on symbolic differentiation and symbolic processing, as implemented in OpenModelica, performed faster.

Memory consumption in the model compilation step was not included in the benchmarks, because of the inherent difficulties in accurately measuring this quantity. Indeed, this measure would have been an interesting addition to the benchmarks presented in this paper, especially since the two methods take different approaches to generate directional derivatives. However, measurements of memory consumption is left for

future work.

6 Acknowledgments

Modelon's contribution to this work was partially funded by Vinnova within the ITEA2 project OPENPROD (dnr: 2010-00068). University of Applied Sciences Bielefeld's contribution to this work was partially funded by The German Ministry BMBF (01IS09029C) within the ITEA2 project OPENPROD. Johan Åkesson acknowledges financial support from Lund Center for Control of Complex systems, funded by the Swedish research council. The authors also would like to acknowledge the kind support from Francesco Casella, who provided the power plant model used for benchmarks.

References

- [1] A. Griewank A. Walther. Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation, Second Edition. SIAM, 2008.
- [2] The Functional Mock-up Interface for Model Exchange 1.0, http://functional-mockup-interface.org/specifications/FMI_for_ModelExchange_v1.0.pdf, January 2010.
- [3] The Functional Mock-up Interface for Co-simulation 1.0, http://functional-mockup-interface.org/specifications/FMI_for_CoSimulation_v1.0.pdf, October 2010.
- [4] R. Tarjan. "Depth-first search and linear graph algorithms." *SIAM J. Computing*, **1:2**, pp. 146–160, 1972.
- [5] D. H. Al-Omari K. E. Sabri. "New graph coloring algorithms." *American Journal of Mathematics and Statistics*, 2006.
- [6] T. F. Coleman J. J. More. "Estimation of sparse Jacobian matrices and graph coloring problems." Society for Industrial and Applied Mathematics, 1983.
- [7] Dürrbaum A., Klier W., Hahn H.: Comparison of Automatic and Symbolic Differentiation in Mathematical Modeling and Computer Simulation of Rigid-Body Systems. In: *Multibody System Dynamics*. Springer Netherlands, 2002.
- [8] Y. B. Gol'dshtein. "Portrait of the inverse of a sparse matrix." *Cybernetics and Systems Analysis*, **28**, pp. 514–519, 1992.
- [9] Braun W, Gallardo Yances S, Link K, Bachmann B. Fast Simulation of Fluid Models with Colored Jacobians . In: *Proceedings of the 9th Modelica Conference*, Munich, Germany, Modelica Association, 2012.
- [10] Braun W, Ochel L, Bachmann B. Symbolically Derived Jacobians Using Automatic Differentiation - Enhancement of the OpenModelica Compiler. In: *Proceedings of the 8th Modelica Conference*, Dresden, Germany, Modelica Association, 2011.
- [11] Casella, F., Donida, D., Åkesson, J. Object-Oriented Modeling and Optimal Control: A Case Study in Power Plant Start-Up. In: *Proceedings of the 18th IFAC World Congress*, Milan, Italy, 2011.

Designing models for online use with Modelica and FMI

Pål Kittilsen^{1,2} Svein Olav Hauger¹ Stein O. Wasbø¹

¹Cybernetica AS, 7038 Trondheim, Norway

²Statoil Research Centre, 7005 Trondheim, Norway

pkit@statoil.com

{svein.o.hauger, stein.wasbo}@cybernetica.no

Abstract

Model-based online applications such as soft-sensing, fault detection or model predictive control require representative models. Basing models on physics has the advantage of naturally describing nonlinear processes and potentially describing a wide range of operating conditions. Implementing adaptivity is essential for online use to avoid model performance degradation over time and to compensate for model imperfection. Requirements for identifiability and observability, numerical robustness and computational speed place an upper limit on model complexity. These considerations motivate that models for online use should be balanced-complexity, physically based with online adaption possible.

Despite potential benefits, the effort required to implement balanced-complexity models, particularly at large scales, may deter their use. This paper presents techniques used in the design of balanced-complexity models. A Modelica-based approach is chosen to reduce implementation effort by interfacing exported Modelica models with application code by means of the generic interface FMI. The suggested approach is demonstrated by parameter estimation for a process of offshore oil production: a subsea well-manifold-pipeline production system.

Keywords: modeling, process control, process models, process simulators, offshore oil and gas production, Modelica, subsea production, multiphase flow, balanced-complexity models, nonlinear model-predictive control, FMI

1 Introduction

In this paper the term *online model* refers to a model that tracks the state of a process over time and is implemented with adaptivity. *Adaptivity* in this paper can refer to either state estimation, parameter estimation,

or both.

Applications that can benefit from online models include online simulators for “what-if” and look-ahead analysis, data reconciliation, soft-sensors, fault detection, advisory decision support systems, (nonlinear-) model predictive control (nMPC) and real-time optimization. Such applications have in common that real-time computations are performed on a model that hopefully represents the process with sufficient accuracy. Evaluating and comparing multiple simulation scenarios internally within real-time requirements place conditions on computational speed. Algorithms that evaluate models at different combinations of inputs, states and parameters place requirements on numerical robustness.

Unless the process is time-invariant and the fitted model matches the process perfectly, the model’s ability to track process states will degrade over time. For industrial processes, both time-variation and model imperfections must be expected, which makes adaptivity a crucial factor in the maintenance of model-based online applications. Adaptivity can also be exploited to simplify aspects of modeling for online use, to be discussed.

Identifiability and observability considerations place limits on how many states and parameters that can be uniquely adapted to a given set of measurements of a process. As a consequence, adapting all the parameters and states that are uncertain or time-varying in complex models will often be an ill-posed problem for the available set of measurements. Some authors have suggested converting full-complexity engineering simulators into online models, see for instance [11], but few references are found in the literature of the use of such models for the online applications listed above.

Balanced-complexity models in this paper refer to models based mainly on physics which are specifically designed to adhere to requirements set by online use

for identifiability, observability, numerical robustness and computational speed. In control literature many references to purpose-built online models are found, some recent applications related to process control and oil and gas applications are; industrial batch process: [14], thin-rim oil reservoirs: [13] and [15], riser slugging in multiphase flows: [9] and drilling: [7].

Balanced-complexity models cited in the literature are usually quite small in scope, and for the applications listed above they typically describe a particular piece of equipment or a specific phenomenon of interest in a subsection of a larger plant. Often such models found in the literature are small-scale, on the order of 10 states or less and are feasible to hand-code. There may be synergies to monitoring and controlling large plants in a unified manner instead of as a series of smaller subsystems, a recent discussion of this idea applied to subsea fields is found in [1]. A balanced-complexity model of such larger systems can have hundreds of states, for instance when modeling an entire offshore processing plant, see [16]. At this scale, balanced-complexity models become challenging to code and maintain manually, and it can be challenging to re-use code and to collaborate on model design. *Large-scale* in this paper refers to balanced-complexity models which attempt to describe large systems, and where challenges related to the scale of the model can potentially deter their use.

Modelica has several advantages that can aid in the synthesis of large-scale balanced-complexity models for online use. First, Modelica is declarative and equation-based, meaning that models are expressed by writing differential and algebraic equations, and Modelica compilers interpret these equations into algorithmic code (usually to the C programming language). Second, Modelica is object-oriented and supports building larger models by connecting smaller sub-models. Third, Modelica supports collecting sub-models into libraries that can be shared, re-used and combined as needed. Fourth, most Modelica environments support exporting models with functional-mock up interface(FMI), to be discussed in Section 3.

An earlier reference to work on interfacing translated Modelica code with online control applications is found in [8]. A reference to a similar vendor-specific approach is found in [6]. Several authors have considered interfacing translated Modelica code with optimization algorithms offline, see for instance [10] and [2] for trajectory planning in power plant control.

This paper is to a large extent motivated by development of nMPC for offshore oil and gas produc-

tion, however much of the discussion is independent of process and application. The excitation resulting from normal operation in offshore oil and gas fields can be very low as documented in [3], and this motivates the use of physical modeling and nNMPC, as this approach has reduced need for excited data, see [5]. Some recent applications of nMPC to smart wells are [12], who used a full reservoir simulator as a process model, and [15] who took a balanced-complexity modeling approach. Earlier references to work on large-scale balanced-complexity modeling for offshore oil and gas production are found in [8], which considered the topside processing system, and in [16] which considered a well-pipeline-riser-processing system.

Despite the widespread use of balanced-complexity models reported in control engineering literature, the idea that models for online use should be purpose-built is not widely accepted by industry practitioners with backgrounds in other engineering disciplines. Motivated by this observation, the first purpose of this paper is to present argumentation for the use of balanced-complexity models and then present techniques used in their design. Secondly, this paper discusses how Modelica can be used to simplify the process of synthesizing large-scale balanced-complexity models and to integrate them in online applications.

The paper is structured as follows: Section 2 outlines techniques for the design of balanced-complexity models. Next, Section 3 discusses techniques for interfacing models written in Modelica with control applications. Section 4 presents a case study of using Modelica to build a large-scale balanced-complexity model of an offshore processing plant for state estimation.

2 Synthesis of balanced-complexity models for online use

2.1 The purpose dictates the model

Modeling is to map a real world object into a simpler representation - in this context, into a set of equations. It is the modeler's choice which of the real object's properties and features the model should mimic. Emphasis on the purpose of the model leads naturally to a set of required model properties. Including details not contributing to fulfilling the model's purpose adds computational load, degrades identifiability and increases challenges of robustness.

Example 1. If the purpose of a model based tool is to control the pressure in a gas tank, it is sufficient to model the pressure with the ideal gas law (or poten-

tially modified with a compressibility factor), lumping all gas components into one pseudo-component. However, if the purpose is to control e.g. the CO₂ fraction, one needs to include a component balance and have at least two components: CO₂ and the ‘remaining’-component.

2.2 Techniques for developing balanced-complexity models

This section introduces some techniques that can be useful for developing balanced-complexity models. The techniques are illustrated with examples from an in-house Modelica library developed for online use (see Section 3):

Adaptivity: Candidate adaptivity parameters have significant influence on the solution, yet are known to be difficult or complicated to model with accuracy and/or are slowly time-varying. Which parameters to adapt is determined by analysis of the equation set, literature and by comparison with real-world data. Adaptivity has the ability to reduce model complexity as it may reduce the need for complex empirical correlations in the equation set.

Example 2. Modeling multiphase flow in pipelines is complex, as key parameters such as pressure drop coefficients and gas-liquid velocity distributions depend on many factors that may be difficult to describe accurately with experimental correlations, and as these parameters may also vary with time. The ratio of gas velocity to liquid velocity in multiphase flow can depend on many factors such as flow-regime, Reynolds-numbers, incline angles or others. By choosing the slip factor, the ratio of gas velocity to liquid velocity, as an adaption parameter the challenge of accurately modeling this ratio is mitigated. As modeling the gas-liquid velocity distribution can be complex and can add to model uncertainty, the resulting online model with adaption in slip ratio need not be less accurate than offline counterparts.

Example 3. Centrifugal compressor models are static and based on compressor maps of polytropic head versus volumetric rate, parameterized in compressor speed. The compressor maps supplied by equipment vendors may be subject to inaccuracies and slow changes over time due to wear and tear. A single adaption parameter is introduced to linearly scale the compressor map.

Thereby inaccuracies and time-varying effects in the compressor can be adjusted for in online compressor models.

Explicit models: Deriving model equations from physics often results in models which are differential-algebraic equations sets (DAEs). Solving such equation sets can be both time consuming and subject to numerical stability issues. It is desirable to re-formulate such models as ordinary differential equation sets (ODEs) to improve numerical speed and stability. Especially implicit algebraic equations requiring dedicated solvers should be avoided. Simple algebraic relations can often be solved by rearranging equations. Artificial dynamic variables can be introduced in more challenging cases to break algebraic loops.

State selection: Another key to avoid implicit equations is to formulate the problem explicitly in terms of states. State variables should be selected so that other dependent properties can be calculated explicitly. This is a common challenge particularly when calculating thermodynamic properties. For instance, if thermodynamic relations are explicit in pressure and temperature, pressures and temperatures should be chosen as states. The Modelica language has support for setting preferred state variables while still formulating derivatives using other variables. A Modelica compiler will automatically differentiate the differential equations in order to change the state variables to the preferred set, see [4].

Smoothing: When models are used in conjunction with optimization algorithms it is important that they are continuous and differentiable. To ensure this property, all equations used must be analyzed with regard to smoothness before use, and where needed, artificial transition functions can be included to enforce smoothness.

Right level of detail: For efficient models, the level of detail for a specific phenomenon in the model should match the importance of that particular phenomenon. As discussed in Section 2.1, phenomena which do not contribute to fulfill the purpose of the model should be left out, illustrated by the example below:

Example 4. A common approach in process simulators is to model hydrocarbon fluids with a multi-component mixture, often with 10 or more

components. The high number of components leads to a large number of thermodynamic state variables. For phase equilibrium calculations the common approach is to use iterative algorithms for solving the resulting equation set.

A multiphase medium in an in-house model library is implemented using a low number of components: The gas phase normally contains only one high and one low molecular weight component. This is sufficient to make any gas mixture with an average molecular weight between the two components. A similar approach is taken with the oil/condensate phases, optionally with a water component to be used if water content in oil/condensate is of interest. In addition to the low number of state variables resulting from this approach, an advantage is that a phase equilibrium in a two component mixture can always be calculated explicitly. This is considered as a sufficient level of detail for the purposes of pressure and level control.

Utilize operational conditions: Knowledge of the operational conditions for which the model is applied can simplify the model considerably. It is unnecessary to include descriptions of operational conditions which will never occur. For example, if it is known that the model will be used for a process with strict temperature control, it will be a good approximation to drop the energy balance and use constant temperature in the model.

Pre-computation of properties: A common model simplification technique is to tabulate complex relations, for instance thermodynamic properties. In this way, complex calculations can be pre-computed, and when used online models can access the ready solutions. If tables are used, attention should be paid to the selection of table interpolation algorithm as to avoid non-smooth derivatives of the interpolated functions. Since searching through large tables is time consuming, simple function approximations is a good alternative.

Data-driven modeling: Data from operation of a process can be used for selecting the right model. Process data with excitations can reveal hints of what model structures can emulate the process. One could either look for a physical phenomenon giving the same response as the data, or consider

introducing a semi-empirical model component which replicates the observed response. For empirical equations, care should be taken when extrapolating.

3 Efficient large-scale modeling by the use of Modelica

The approach to efficient large-scale modeling considered in this paper is outlined in Figure 1. The approach is based on implementing the *Functional-mock up interface (FMI)*¹ in software used in online control applications. An FMI-standard model component is shared as a *functional mock-up unit (FMU)*.

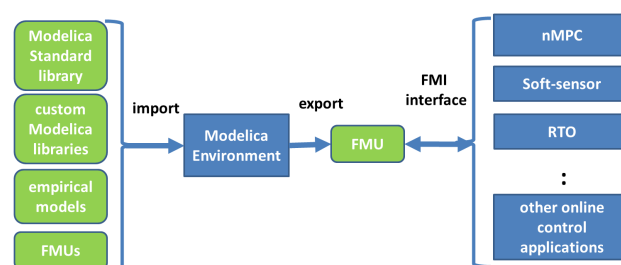


Figure 1: Flow of information between models (rounded edges) and applications (straight edges).

Since the translation from Modelica to FMI is done by a compiler, and as all low-level code to interface model and online application is model-independent and re-usable, the transition from Modelica to online applications can be made in a matter of minutes. This framework supports an iterative modeling work flow, as repeating the conversion from model to application multiple times is not workload-intensive.

Aside from the advantages of Modelica listed in Section 1, a benefit of designing models in a Modelica environment is that sub-modules can be imported from multiple external sources. The ability to import modules as FMUs means that the process owner, equipment suppliers or others can supply proprietary models as pre-compiled FMUs. This also opens an avenue for suppliers of process simulators to export their models seamlessly into control applications, provided they implement support for export of models as FMUs. For the reasons mentioned in Section 1, it will still be advantageous for such models to be designed with the techniques discussed in Section 2.

When designing large-scale models, it is often desirable to model selected subsystems or components

¹see <http://www.modelisar.com/>

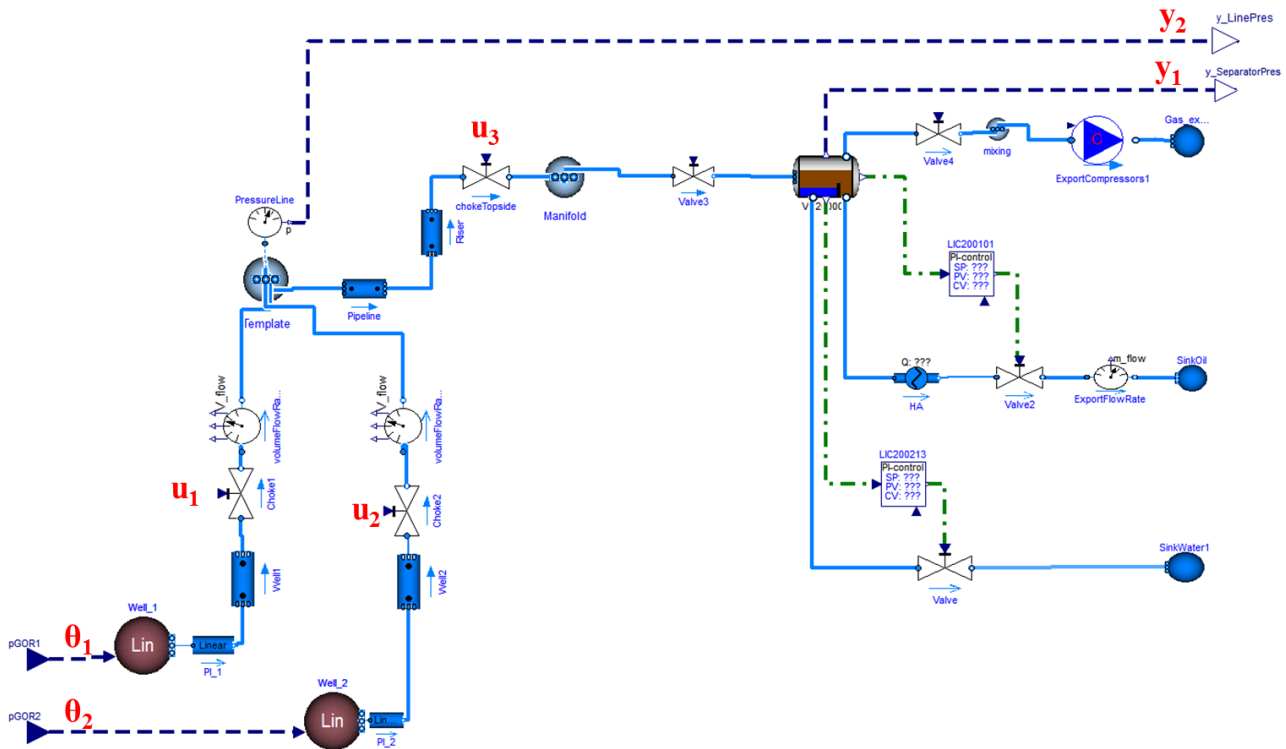


Figure 2: Overview subsea-pipeline-riser-separator system as implemented in DYMOLA, with piping (solid), handles to the estimator via FMI (dashed), and PI-control (dashdot).

of the larger system using empirical models, for instance fitted curves or state-space models inferred from data. An efficient manner of incorporating such sub-models in a larger Modelica-based framework is to express empirical models in the Modelica language. Exporting empirical models in Modelica-form is a task that can be automated by software for system identification. The modular buildup of Modelica allows such exported models to be seamlessly integrated with physics-based Modelica models.

4 Case-study: Estimation of gas-oil ratio in offshore oil and gas production

The aim of this case study is to illustrate that a large-scale balanced-complexity model which has been designed along the principles outlined in Section 2 can be implemented efficiently by the methods outlined in Section 3. The case considered is *stylized* in that for demonstration purposes, the estimator used has a relatively low number of fitted parameters and measurements.

The system considered is shown in Figure 2, and consists of the joint production of oil, gas and water

from two different wells. The fluids from the wells are mixed in a subsea template before traveling along a horizontal pipeline, through a vertical riser, into a topside manifold before reaching the topside processing plant. The production rates from the two wells are not measured directly, yet these flow rates are of great interest as they determine production revenues and the feed rates to which the process plant must adapt.

There is a significant static pressure drop from the reservoir to the sea bed (elevation often being of order thousands of meters) and from the sea bed to the floating production unit (elevation often of order hundreds of meters). The static pressure drop depends on the ratio of gas-to-liquid, and as the proportion of total production that is water is often fairly constant, it should be possible to infer about the gas-oil ratio by modeling its relationship to pressure in well and pipeline. Since pressure in the pipeline depends on the settings of chokes on each well and upstream of the separator, these chokes must also be modeled.

A typical full-complexity multiphase pipe flow simulator could for the well-pipeline-separator system considered have hundreds or thousands of control volumes, and a full-complexity thermodynamic model could have on the order of 20 states for each control volume. Thermodynamic relations would in a full-

complexity model depend on implicit relations, and a large number of different empirical closure relations for different conditions would be used in multiphase flow models.

From our perspective such a full-complexity model would be unsuitable for the purposes of estimating gas-oil ratio online, due to the issues mentioned in Section 1.

4.1 Modeling

Modules from an in-house Modelica library were used and put together with the aim of finding the right level of detail to achieve the desired goal of estimating gas-oil ratio. It was elected to model flow as a two-phase flow, lumping oil and water flows into a single liquid flow. Modules describing wells, horizontal and vertical pipelines and chokes were combined to create the large-scale model. Each of these modules were originally designed by combining first-principles with empirical closure relations from the literature that were revised for simplicity, to obtain smoothness and to avoid implicit relations. The number of different closure relations was kept as low as possible, and the resulting models were validated module-for-module against real-world data. The modules include handles for introducing adaptivity as needed through adjustable parameters such as gas-liquid velocity ratios, valve coefficients and friction factors. Adapting the mentioned parameters was omitted here for simplicity.

Some examples of the balanced-complexity principles in the current case-study follows:

- **Exclude flashing.** From experience and analysis of real-world data similar to this case, the flashing (evaporation of dissolved gas in the oil) as the pressure drops in the pipeline is not expected to be significant relative to amount of free gas. Excluding flashing from the model was therefore judged to be the right level of detail.
- **Few discrete mixing volumes.** Riser and pipeline models are finite-volume spatial discretizations of the underlying partial-differential flow equations, and the number of discrete volumes for each of these modules are design parameters that the user should select at design while evaluating resulting model accuracy. It is our experience that no fine discretization is required for estimators such as considered here to work. Lumping pipeline submodels into two or even just one volume is often found to be the right level of detail. For each volume in each sub-model, a

mass-balance equation is formulated and a simplified thermodynamic relation with a low number of components that is smooth and explicit, as described in Example 4, was used.

- **Limiting the scope of the model.** The three-phase separator model uses a thermodynamic equilibrium equation for flashing/vaporization, in combination with a mass balance that takes in account separator geometry. Since the estimator considers the portion of the offshore oil and gas system spanning from wells to the separator, it was not considered necessary to model further downstream process equipment for the desired estimator, motivated by the concept of the purpose dictating the model.

All the models were expressed in equation-form in the Modelica language, and the translation capabilities of DYMOLA were used to convert this equation-based model into an imperative, C-language code that is suitable for online use. The model shown as drawn by DYMOLA is shown in Figure 2. That the imperative code of the model is generated rather than hand-coded directly is useful for iteratively deciding the right level of detail in the model. The degree of model detail is easily adjustable in the high-level, modular, equation based language Modelica, from which multiple estimators based on different low-level implementations of the model in C can be compared.

4.2 Estimation

Simulations were done for a model with only a single node for pipeline and riser. The resulting model has 48 states, DYMOLA choosing five states (pressure + 4 component mass fractions) for each of the nodes: well 1, well 2, subsea manifold, pipeline, riser, topside manifold and inlet separator.

Pressures at the topside separator (y_1) and subsea manifold (y_2) were chosen as outputs. Parameters were chosen as gas-oil ratios of well 1 (θ_1) and well 2 (θ_2). Choke openings of valves on well 1 (u_1), well 2 (u_2) and the topside valve (u_3) are varied during the simulation. The estimator used is a recursive Extended Kalman Filter (EKF). The model was implemented in Modelica, compiled as an FMU using DYMOLA, and interfaced with a generic and re-usable recursive Extended Kalman Filter (EKF).

The dataset considered is synthetic, generated by simulating a copy of the model where the gas-oil ratios of both wells were set equal to 811. Noise of 2% of average amplitude was added to both pressures.

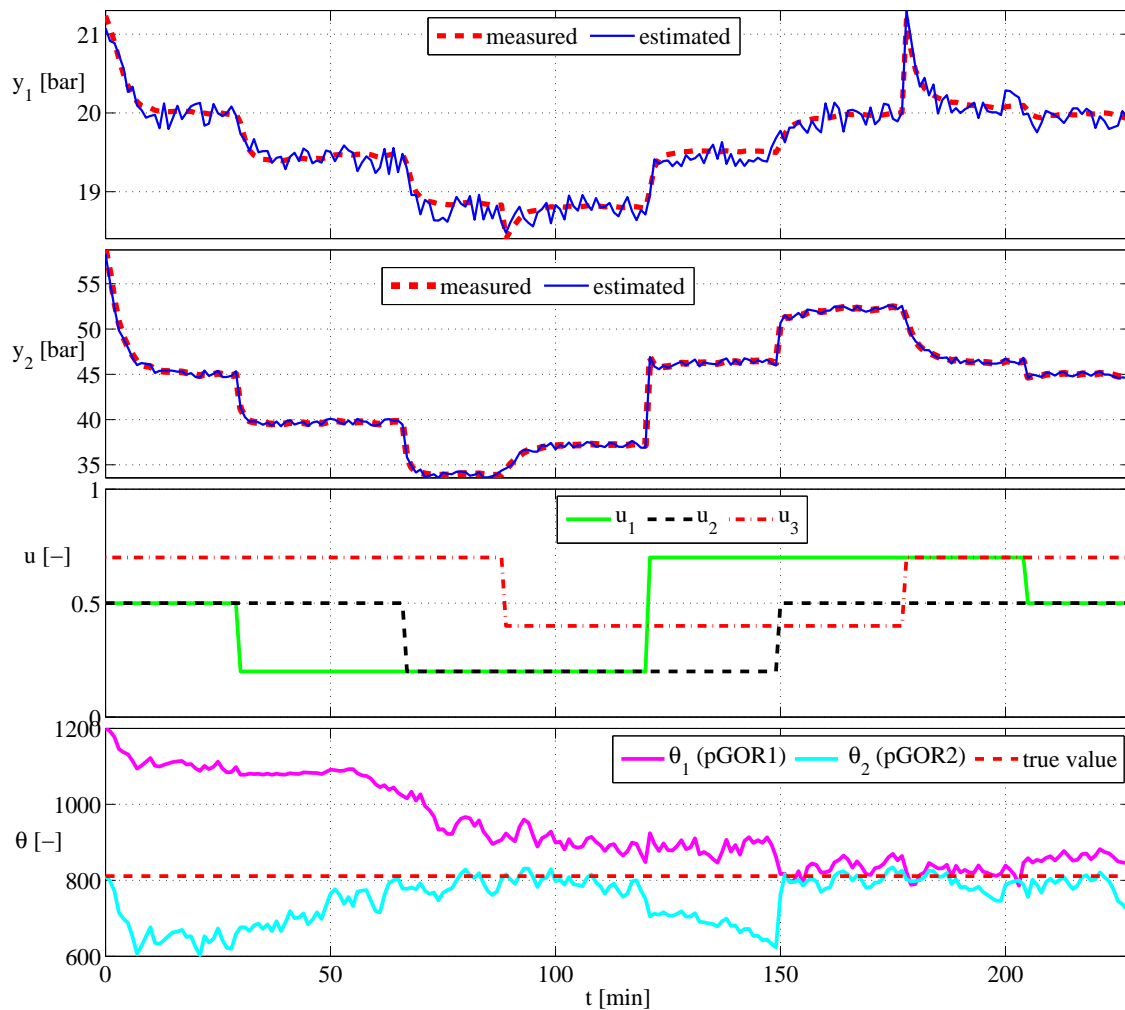


Figure 3: Simulation results. Top subplot shows measured and estimated separator pressures. Subplot 2 shows measured and estimated line pressures. Subplot 3 shows relative choke opening of wells 1 and 2 (u_1 and u_2) and of the topside valve (u_3). Subplot 4 shows recursive estimates of gas-oil ratios of wells 1 and 2 compared with the true value.

4.3 Simulations

Estimated and measured pressures and estimated gas-oil ratios for wells 1 and 2 are shown in Figure 3. The initial estimate for the gas-oil ratio of well 1 was set to 1200, while the actual gas-oil ratio for both wells is 811. The inaccurate initial estimate of gas-oil ratio resulted in an offset between measured and modeled pressures, which the estimator attempts to correct during simulation. The excitation shown in Figure 3 made it possible to uniquely determine gas-oil ratios for both wells from the data set, and as the simulation progresses, the estimated gas-oil ratios move toward the real value of 811.

4.4 Discussion

The main contribution of this case study is the technology and workflow used to implement an online model including Kalman Filter estimators. The solution was implemented in a low-level language suitable for online use, yet no line of low-level code was manually written. The model used has 48 states, and manually implementing low-level model code would be a challenging task already at this scale if you consider that modeling requires several design iterations, collaboration among multiple designers, code-reuse and code validation. Our experience indicates that the approach could accommodate working efficiently on

much larger models as well.

5 Conclusion

Balanced-complexity modeling is an approach to bring physics-based models online while adhering to requirements for online use. Modelica and FMI have advantages that aid the development of such systems: efficient model development; reuse of models; and efficient integration with other software. By calling attention to this topic it is hoped for an increased recognition for online applications with purpose-built models developed with Modelica and FMI.

References

- [1] B. Bringedal, E. Storakaas, M. Dalsmo, M. Aarset, and H. M. With. Recent developments in control and monitoring of remote subsea fields. In *SPE Intelligent Energy Conference and Exhibition*, Utrecht, The Netherlands, 2010.
- [2] F. Casella, F. Donida, and J. Akesson. Object-oriented modeling and optimal control: A case study in powerplant start-up. In *Proc. 18th IFAC World Congress, Milano, Italy*, volume 18, 2011.
- [3] S. Elgsæter, O. Slupphaug, and T. A. Johansen. Challenges in parameter estimation of models for offshore oil and gas production optimization. In *International Petroleum Technology Conference*, Dubai, 2007.
- [4] H. Elmqvist, H. Tummescheit, and M. Otter. Object-oriented modeling of thermo-fluid systems. *Proceedings of 3rd Int. Modelica Conference*, pages 269–286, 2003.
- [5] B. A. Foss and T. S. Schei. Putting nonlinear model predictive control into use. In *Assessment and Future Directions Nonlinear Model Predictive Control*, LNCIS 358, pages 407–417. Springer Verlag, 2007.
- [6] R. Franke, B. S. Babji, M. Antoine, and A. Isaksson. Model-based online applications in the abb dynamic optimization framework. In *Modelica'2008*, Bielefeld, Germany, 2008.
- [7] J. M. Godhavn, A. Pavlov, G. O. Kaasa, and N. L. Rolland. Drilling seeking automatic control solutions. In *Proc. 18th IFAC World Congress, Milano, Italy*, volume 18, Milano, Italy, 2011.
- [8] L. Imsland, P. Kittilsen, and T. S. Schei. Model-based optimizing control and estimation using modelica models. In *Modelica'2008*, Bielefeld, Germany, 2008.
- [9] E. Jahanashahi and S. Skogestad. Simplified dynamical models for control of severe slugging in multiphase risers. In *Proc. 18th IFAC World Congress, Milano, Italy*, volume 18, Milano, Italy, 2011.
- [10] K. Krueger, M. Rode, R. Franke, and B. A. Foss. Optimization of boiler start-up using a nonlinear boiler model and hard constraints. *Energy*, 29(12-15):2239–2251, 2004.
- [11] S. McArdle, D. Cameron, and K. Meyer. The life cycle simulator: From concept to commissioning... and beyond. In *SPE Intelligent Energy Conference and Exhibition*, pages 246–267, Utrecht, The Netherlands, 2010.
- [12] P. Meum, P. Tøndel, J. M. Godhavn, and O. M. Aaamo. Optimization of smart well production through nonlinear model predictive control. In *Intelligent Energy Conference and Exhibition*, Amsterdam, The Netherlands, 2008.
- [13] A. Mjaavatten, Robert Aasheim, Steinar Saelid, and Oddvar Gronning. Model for gas coning and rate-dependent gas/oil ratio in an oil-rim reservoir. *SPE Reservoir Evaluation & Engineering*, 11(5), 2008.
- [14] Z. K. Nagy, B. Mahn, R. Franke, and F. Allgöwer. Nonlinear model predictive control of batch processes: an industrial case study. In *Proc. 16th IFAC World Congress, Prague, Czech Republic*, volume 16, 2005.
- [15] R. van der Linden and A. Leemhuis. The use of model predictive control for asset production optimization: Application to a thin-rim oil field case. In *SPE Annual Technical Conference and Exhibition*, Florence, Italy, 2010.
- [16] A. Willersrud, L. Imsland, S. O. Hauger, and P. Kittilsen. Short-term production optimization of offshore oil and gas production using nonlinear model predictive control. In *Proc. 18th IFAC World Congress, Milano, Italy*, volume 18, Milano, Italy, 2011.

Co-simulation with communication step size control in an FMI compatible master algorithm

Tom Schierz* Martin Arnold* Christoph Clauß#

* Martin Luther University Halle-Wittenberg
NWF II - Institute of Mathematics
D - 06099 Halle (Saale), Germany
martin.arnold@mathematik.uni-halle.de
tom.schierz@mathematik.uni-halle.de

Fraunhofer Institute for Integrated Circuits IIS
Design Automation Division EAS
Zeunerstr. 38
D - 01069 Dresden, Germany
christoph.clauss@eas.iis.fraunhofer.de

Abstract

Complex multi-disciplinary models in system dynamics are typically composed of subsystems. This modular structure of the model reflects the modular structure of complex engineering systems. In industrial applications, the individual subsystems are often modeled separately in different mono-disciplinary simulation tools. The Functional Mock-Up Interface (FMI) provides an interface standard for coupling physical models from different domains and addresses problems like export and import of model components in industrial simulation tools (FMI for Model Exchange) and the standardization of co-simulation interfaces in nonlinear system dynamics (FMI for Co-Simulation), see [8]. In November 2011, the third β -version of FMI for Model Exchange and Co-Simulation v2.0 was released [13] that supports advanced numerical techniques in co-simulation like higher order extrapolation and interpolation of subsystem inputs, step size control including step rejection and Jacobian based linearly implicit stabilization techniques. Well known industrial simulation tools for applied dynamics support Version 1.0 of this standard and plan to support the forthcoming Version 2.0 in the near future, see the “Tools” tab of website [8] for up-to-date information. The renewed interest in algorithmic and numerical aspects of co-simulation inspired some new investigations on error estimation and stabilization techniques in FMI for Model Exchange and Co-Simulation v2.0 compatible co-simulation environments. The present paper extends recent results from [3] on reliable error estimation and communication step size control in the framework of FMI for Model Exchange and Co-Simulation v2.0. Based on a strict mathematical analysis, we study the asymptotic behaviour of the local error and two error estimates that may be used to

adapt the communication step size automatically to the changing solution behaviour during time integration. These theoretical results are illustrated by numerical tests for a (linear) quarter car model and provide a basis for future investigations with more complex coupled engineering systems.

Keywords: FMI; error estimation; step size control

1 Introduction

Co-simulation is a rather general approach to the simulation of coupled technical systems and coupled physical phenomena in engineering with focus on instationary (time-dependent) problems. From the mathematical viewpoint, co-simulation results in a class of time integration methods for coupled systems which are described by time dependent ordinary differential equations (ODE) or differential algebraic equations (DAE). In that context, we consider $r \geq 2$ coupled subsystems in nonlinear state-space form

$$\left. \begin{aligned} \dot{\mathbf{x}}_i &= \mathbf{f}_i(t, \mathbf{x}_i, \mathbf{u}_i), \\ \mathbf{y}_i &= \mathbf{g}_i(t, \mathbf{x}_i, \mathbf{u}_i), \end{aligned} \right\} \quad i = 1, \dots, r, \quad t \in [t_{\text{start}}, t_{\text{stop}}] \quad (1a)$$

with the state vectors \mathbf{x}_i , inputs \mathbf{u}_i and outputs \mathbf{y}_i . The subsystems are coupled by input-output relations

$$\mathbf{u}_i = \mathbf{c}_i(\mathbf{y}_1, \dots, \mathbf{y}_{i-1}, \mathbf{y}_{i+1}, \dots, \mathbf{y}_r), \quad (i = 1, \dots, r), \quad (1b)$$

see [12]. Summarizing all components in vector form, we get a coupled system in the more compact form

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{f}(t, \mathbf{x}, \mathbf{u}), \\ \mathbf{y} &= \mathbf{g}(t, \mathbf{x}, \mathbf{u}), \quad \mathbf{u} = \mathbf{c}(\mathbf{y}), \end{aligned} \quad (2)$$

with $\mathbf{x} = (\mathbf{x}_1^T, \dots, \mathbf{x}_r^T)^T, \dots$

The simulation time interval is split by a grid of *communication points* $t_{\text{start}} = T_0 < T_1 < \dots < T_N = t_{\text{stop}}$, where the data exchange between the subsystems is restricted to these discrete points. In each *communication step* $T_n \rightarrow T_n + H_n =: T_{n+1}$, ($n = 0, \dots, N-1$), with *communication step size* H_n , the subsystems are solved separately and independently from each other.

Since the update of the inputs \mathbf{u} is restricted to the time discrete communication points, these terms have to be approximated by signal extrapolation for the current communication step $T_n \rightarrow T_{n+1}$

$$\bar{\mathbf{u}}(t) \approx \mathbf{u}(t), \quad t \in [T_n, T_{n+1}],$$

where the approximation $\bar{\mathbf{u}}(t)$, is based on information $\mathbf{u}(T_{n-i})$, ($i = 0, \dots, k$) from $k+1$ previous communication points.

This leads to the new coupled problem for $t \in [T_n, T_n + H_n]$

$$\begin{aligned} \dot{\bar{\mathbf{x}}}(t) &= \mathbf{f}(t, \bar{\mathbf{x}}, \bar{\mathbf{u}}), \\ \bar{\mathbf{y}}(t) &= \mathbf{g}(t, \bar{\mathbf{x}}, \bar{\mathbf{u}}). \end{aligned} \quad (3)$$

The coupled system (3) is composed of subsystems that are solved separately from each other, where we get the numerical solution $\bar{\mathbf{x}}(t) \approx \mathbf{x}(t)$ and $\bar{\mathbf{y}}(t) \approx \mathbf{y}(t)$, ($t \in [T_n, T_n + H_n]$). Typically, different time integration methods and / or different (*micro*) step sizes are used in the individual subsystems resulting in a multi-method and / or multi-rate method for the coupled system.

Following Jackson [10], the time integration methods to solve the coupled system (3) are called *modular* to underline the modular structure of the approach.

In practical applications, usually constant communication step sizes $H_n := H$ with $n = 0, \dots, N-1$ are used, since the simulation results and the behavior of the underlying integration methods can reliably be controlled. Further gains in efficiency and robustness of numerical co-simulation techniques are expected by variable communication step sizes $H_n := T_{n+1} - T_n$ that are adapted automatically to the solution behavior (communication step size control).

2 Estimation of the local error

For a reliable communication step size control an error estimate **EST** for the local error of the numerical solution is needed and is compared to user defined error bounds (tolerances), since for coupled systems (2) without algebraic loops the global error is bounded by a multiple of the maximum local error [3].

Richardson extrapolation is a time-consuming but reliable way to estimate local errors in ODE and DAE

time integration and considers two (small) consecutive communication steps of size $H_n = H_{n+1} = H$ from $T_n \rightarrow T_n + H =: T_{n+1}$ and $T_{n+1} \rightarrow T_n + 2H =: T_{n+2}$. To get an estimate for the local error, the solution of these two steps is compared with a second numerical solution that is computed for a (large) communication step $T_n \rightarrow T_n + 2H =: T_{n+2}$. Substantial savings of computing time result from an algorithmic modification of the Richardson extrapolation that is tailored to the FMI co-simulation framework. Both approaches will be studied theoretically (by an asymptotic error analysis) as well as practically (by numerical tests for a quarter car benchmark problem). For the theoretical analysis, we neglect the discretization errors in the subsystems that should be kept small in a practical implementation by appropriate settings of error tolerances.

To keep the notation compact, we restrict the theoretical analysis of the local error estimates to pure polynomial signal extrapolation in all subsystems [3]. Consider the polynomial $\bar{\mathbf{u}}(t)$, that interpolates the analytical solution $\mathbf{u}(t)$ of (2) at the $k+1$ equidistant previous communication points T_{n-i} , ($i = 0, \dots, k$). The approximation error of this interpolating polynomial $\bar{\mathbf{u}}(t)$ is for all $t \in [T_n, T_n + H]$ bounded by [7]

$$\begin{aligned} \bar{\mathbf{u}}(t) - \mathbf{u}(t) &= -\frac{\mathbf{u}^{(k+1)}(T_n)}{(k+1)!} \prod_{i=0}^k (t - T_{n-i}) \\ &\quad + \mathcal{O}(H^{k+2}). \end{aligned} \quad (4)$$

In the first (small) communication step $T_n \rightarrow T_n + H$ (first step of the error estimation by Richardson extrapolation) we have to solve system (3) starting from $\bar{\mathbf{x}}(T_n) = \mathbf{x}(T_n)$. Linearization shows that the error in the state and output vectors satisfy

$$\begin{aligned} \bar{\mathbf{x}}(t) - \mathbf{x}(t) &= \mathbf{A}_n(\bar{\mathbf{x}}(t) - \mathbf{x}(t)) + \mathbf{B}_n(\bar{\mathbf{u}}(t) - \mathbf{u}(t)) + \\ &\quad + \mathcal{O}(H^{k+2}), \\ \bar{\mathbf{y}}(t) - \mathbf{y}(t) &= \mathbf{C}_n(\bar{\mathbf{x}}(t) - \mathbf{x}(t)) + \mathbf{D}_n(\bar{\mathbf{u}}(t) - \mathbf{u}(t)) + \\ &\quad + \mathcal{O}(H^{k+2}), \end{aligned}$$

where the system matrices \mathbf{A}_n , \mathbf{B}_n , \mathbf{C}_n , \mathbf{D}_n denote the Jacobians \mathbf{f}_x , \mathbf{f}_u , \mathbf{g}_x , \mathbf{g}_u evaluated at $\mathbf{x} = \mathbf{x}(T_n)$, $\mathbf{u} = \mathbf{u}(T_n)$. Up to higher order terms, the error in the state vector is given by the solution of a linear time invariant system for $t \in [T_n, T_n + H_n]$ and may be expressed as

$$\begin{aligned} \bar{\mathbf{x}}(t) - \mathbf{x}(t) &= \exp(\mathbf{A}_n(t - T_n)) \overbrace{(\bar{\mathbf{x}}(T_n) - \mathbf{x}(T_n))}^{=0} \\ &\quad + \int_{T_n}^t \exp(\mathbf{A}_n(t - s)) \mathbf{B}_n(\bar{\mathbf{u}}(s) - \mathbf{u}(s)) ds \\ &\quad + \mathcal{O}(H^{k+3}). \end{aligned} \quad (5)$$

With (4) and (5) and the substitution $\sigma := (s - T_n)/H$, $Hd\sigma = ds$ the error of the output vector is

$$\begin{aligned} \bar{\mathbf{y}}(T_{n+1}) - \mathbf{y}(T_{n+1}) = & \\ & - \underbrace{\mathbf{C}_n \mathbf{B}_n \frac{\mathbf{u}^{(k+1)}(T_n)}{(k+1)!} \int_0^1 \prod_{\iota=0}^k (\sigma + \iota) d\sigma}_{=: \gamma_k} H^{k+2} \\ & - \underbrace{\mathbf{D}_n \mathbf{u}^{(k+1)}(T_n)}_{=: \delta_k} H^{k+1} + \mathcal{O}(c_D H^{k+2} + H^{k+3}) \end{aligned}$$

with constant c_D , which is set to $c_D = 0$ if $\partial \mathbf{g} / \partial \mathbf{u} \equiv 0$ and $c_D = 1$ otherwise.

In the next (small) communication step $T_n + H \rightarrow T_n + 2H$ the input function is substituted by a polynomial $\bar{\mathbf{u}}(t)$ that interpolates $\mathbf{c}(\bar{\mathbf{y}})$ at $t = T_n + H$ and $\mathbf{u}(t)$ at $t = T_{n-1}$, ($\iota = 0, \dots, k-1$).

The error in the output vector $\bar{\bar{\mathbf{y}}}(T_{n+2})$ is then given by

$$\begin{aligned} \bar{\bar{\mathbf{y}}}(T_{n+2}) - \mathbf{y}(T_{n+2}) = & \\ & - 2\gamma_k H^{k+2} \\ & - \left(\delta_k + (k+1) \mathbf{D}_n \mathbf{L}_n \delta_k \right) H^{k+1} \quad (6) \\ & + \mathcal{O}(c_D H^{k+2} + H^{k+3}) \end{aligned}$$

with $\mathbf{L}_n = \partial \mathbf{c} / \partial \mathbf{y}(T_n)$.

For error estimation by Richardson extrapolation, we consider in time interval $[T_n, T_n + 2H]$ a second numerical solution for the output vector $\tilde{\mathbf{y}}(T_{n+2})$ and input function $\tilde{\mathbf{u}}(t)$ that is defined by the interpolation polynomial for data points $(T_{n-2\iota}, \mathbf{u}(T_{n-2\iota}))$, ($\iota = 0, \dots, k$). Analogously to the first step we get the error

$$\begin{aligned} \tilde{\mathbf{y}}(T_{n+2}) - \mathbf{y}(T_{n+2}) = & - \gamma_k (2H)^{k+2} - \delta_k (2H)^{k+1} \\ & + \mathcal{O}(c_D H^{k+2} + H^{k+3}). \quad (7) \end{aligned}$$

In ODE and DAE time integration, the comparison of the numerical results for a double-step with (small) step size H and a single (large) step with step size $2H$ allows to estimate the leading term of the local error [9]. For modular time integration, this error estimate is given by [11]

$$\mathbf{EST}_{\text{Rich}} := \frac{\bar{\bar{\mathbf{y}}}(T_{n+2}) - \tilde{\mathbf{y}}(T_{n+2})}{(1 - 2^{k+1})}.$$

The comparison of (6) and (7) shows

$$\begin{aligned} \bar{\bar{\mathbf{y}}}(T_{n+2}) - \mathbf{y}(T_{n+2}) = & \mathbf{EST}_{\text{Rich}} \\ & + \frac{(k+1)2^{k+1}}{(1 - 2^{k+1})} \mathbf{D}_n \mathbf{L}_n \delta_k H^{k+1} \\ & + \mathcal{O}(c_D H^{k+2} + H^{k+3}). \end{aligned}$$

Here we can see, that in the context of co-simulation, Richardson extrapolation may give asymptotically wrong results if $\mathbf{D}_n \mathbf{L}_n \delta_k \neq 0$, i. e., for coupled systems with direct feed-through in at least one subsystem. If there are no subsystems with direct feed-through ($\partial \mathbf{g}_j / \partial \mathbf{u}_j \equiv 0$, ($j = 1, \dots, r$)), $\mathbf{EST}_{\text{Rich}}$ reproduces all components of the local error in the output variables correctly up to higher order terms.

In the ITEA2 project MODELISAR, several modifications of error estimate $\mathbf{EST}_{\text{Rich}}$ were tested [15] to reduce the large extra effort for computing $\bar{\mathbf{y}}$. Here we use $\bar{\mathbf{u}}(t) = \tilde{\mathbf{u}}(t)$ for $t \in [T_n, T_n + H]$ such that the intermediate results $\bar{\mathbf{x}}(T_{n+1})$ and $\tilde{\mathbf{x}}(T_{n+1})$ coincide. From the view point of numerical efficiency, it would be favorable to use one and the same approximation $\bar{\mathbf{u}}(t)$ of the input function $\mathbf{u}(t)$ for both numerical solutions in the first communication step $T_n \rightarrow T_n + H$ and to restrict the use of different input functions to the second communication step, i. e., to $t \in [T_{n+1}, T_{n+2}]$.

In that way, co-simulation may proceed with a large communication step $T_n \rightarrow T_n + 2H$ of size $2H$ that is temporarily interrupted at $t = T_{n+1}$ to provide input data $\bar{\mathbf{y}}(T_{n+1})$ and $\mathbf{c}(\bar{\mathbf{y}}(T_{n+1}))$ for the second numerical solution to be used for error estimation. Alternatively, a small communication step $T_n \rightarrow T_n + H$ may be completed in the classical way and the two different numerical solutions on time interval $[T_{n+1}, T_{n+2}]$ are evaluated in parallel. With this second strategy, no subsystem solver has to go backward in time and the practical implementation might be simplified.

According to [3] the error estimate is then given by

$$\mathbf{EST}_{\text{mod}} := \frac{\bar{\bar{\mathbf{y}}}(T_{n+2}) - \bar{\mathbf{y}}(T_{n+2})}{(1 - p_k)},$$

where p_k is given for constant ($k = 0$), linear ($k = 1$) and quadratic ($k = 2$) extrapolation by

$$\begin{aligned} p_0 &= 2, \\ p_1 &= 14/5, \\ p_2 &= 32/9. \end{aligned}$$

To demonstrate the estimation of the local error and the communication step size control in co-simulation, we use a low dimensional linear benchmark problem. The *quarter car model* which goes along a road profile is defined by a one-dimensional oscillator with two mass points m_c and m_w for the chassis and the wheel. Both masses are coupled by a spring-damper element with spring constant k_c and damping constant d_c , see Figure 1. Moreover, the connection between wheel and road is represented by a spring-damper element

with constants k_w and d_w . For typical parameter values (see below), the eigenfrequency of subsystem wheel is ten times larger than the one of subsystem chassis.

The displacements of the mass points are given by η_c , η_w and the profile of the road is defined by a time dependent function $z(t)$. For numerical tests we use the following parameter configuration:

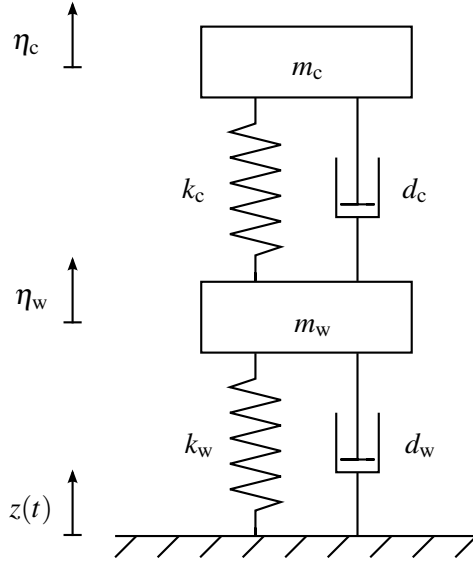


Figure 1: Quarter car model.

wheel mass	$m_w = 40 \text{ kg}$,
chassis mass	$m_c = 400 \text{ kg}$,
wheel spring	$k_w = 150000 \text{ Nm}^{-1}$,
chassis spring	$k_c = 15000 \text{ Nm}^{-1}$,
wheel damper	$d_w = 0 \text{ Nsm}^{-1}$,
chassis damper	$d_c = 1000 \text{ Nsm}^{-1}$,

initial values at $t = 0$

$$\begin{aligned}\eta_c(0) &= 0 \\ \dot{\eta}_c(0) &= 0 \\ \eta_w(0) &= 0 \\ \dot{\eta}_w(0) &= 0\end{aligned}$$

and excitation function

$$z(t) := \begin{cases} 0, & t < 0, \\ 0.1, & t \geq 0 \end{cases}$$

to get the system in motion. The equations of motion are given by

$$\begin{aligned}m_c \ddot{\eta}_c(t) &= k_c(\eta_w(t) - \eta_c(t)) + d_c(\dot{\eta}_w(t) - \dot{\eta}_c(t)), \\ m_w \ddot{\eta}_w(t) &= k_w(z(t) - \eta_w(t)) + d_w(\dot{z}(t) - \dot{\eta}_w(t)) \\ &\quad - k_c(\eta_w(t) - \eta_c(t)) - d_c(\dot{\eta}_w(t) - \dot{\eta}_c(t)).\end{aligned}$$

For co-simulation this system is split into two subsystems (chassis and wheel) of the form

$$\begin{aligned}\dot{\mathbf{x}}_1 &= \mathbf{f}_1(\mathbf{x}_1, \mathbf{u}_1), & \dot{\mathbf{x}}_2 &= \mathbf{f}_2(\mathbf{x}_2, \mathbf{u}_2), \\ \mathbf{y}_1 &= \mathbf{g}_1(\mathbf{x}_1, \mathbf{u}_1), & \mathbf{y}_2 &= \mathbf{g}_2(\mathbf{x}_2, \mathbf{u}_2), \\ \mathbf{u}_1 &= \mathbf{y}_2, & \mathbf{u}_2 &= \mathbf{y}_1\end{aligned}$$

with the state vectors $\mathbf{x}_1 = (\eta_c, \dot{\eta}_c)^T$ and $\mathbf{x}_2 = (\eta_w, \dot{\eta}_w)^T$, right hand sides of the subsystems $\mathbf{f}_1 = (\dot{\eta}_c, \ddot{\eta}_c)^T$, $\mathbf{f}_2 = (\dot{\eta}_w, \ddot{\eta}_w)^T$, inputs \mathbf{u}_1 , \mathbf{u}_2 and outputs \mathbf{y}_1 , \mathbf{y}_2 . We consider a displacement-displacement and a displacement-force coupling [5]. For the displacement-displacement coupling the input and output vectors are given by

$$\begin{aligned}\mathbf{u}_2 &= \mathbf{y}_1 = \mathbf{x}_1, \\ \mathbf{u}_1 &= \mathbf{y}_2 = \mathbf{x}_2.\end{aligned}$$

In the case of displacement-force coupling, the output of the second subsystem (wheel) is defined by the coupling force of the spring-damper element between the two masses m_c and m_w which is also the input of the other subsystem (chassis):

$$\begin{aligned}\mathbf{u}_2 &= \mathbf{y}_1 = (\eta_c, \dot{\eta}_c)^T, \\ \mathbf{u}_1 &= \mathbf{y}_2 = k_c(\eta_w(t) - \eta_c(t)) + d_c(\dot{\eta}_w(t) - \dot{\eta}_c(t)).\end{aligned}$$

In that case, the subsystem wheel has a direct feed-through of its input, $\partial \mathbf{g}_2 / \partial \mathbf{u}_2 \neq \mathbf{0}$.

Figs. 2 and 3 show the local errors for the quarter car benchmark and illustrate that the new error estimate $\mathbf{EST}_{\text{mod}}$ is as reliable as the classical estimate $\mathbf{EST}_{\text{Rich}}$.

3 Variable communication step sizes

The communication step size control in co-simulation is an extension of the step size control in classical time integration methods for ODEs [9].

In the context of co-simulation, the error estimator \mathbf{EST} should estimate in each consecutive communication step $T_n \rightarrow T_{n+1} \rightarrow T_{n+2}$ all errors in the slave outputs $\bar{\mathbf{y}}_{n+2} := \bar{\mathbf{y}}(T_{n+2})$, that result from the solution of (3) by a numerical time integration method with approximated slave inputs $\bar{\mathbf{u}}(t)$, $t \in [T_n, T_{n+1}]$ and $\bar{\mathbf{u}}(t)$, $t \in [T_{n+1}, T_{n+2}]$. We consider the scalar error indicator

$$\text{ERR} := \sqrt{\frac{1}{m} \sum_{j=1}^m \left(\frac{\text{EST}_j}{\text{ATOL}_j + \text{RTOL}_j |\bar{\mathbf{y}}_{n+2}^j|} \right)^2} \quad (8)$$

with the vector

$$\bar{\mathbf{y}}_{n+2} := (\bar{\mathbf{y}}_{1,n+2}^T, \dots, \bar{\mathbf{y}}_{r,n+2}^T)^T \in \mathbb{R}^m,$$

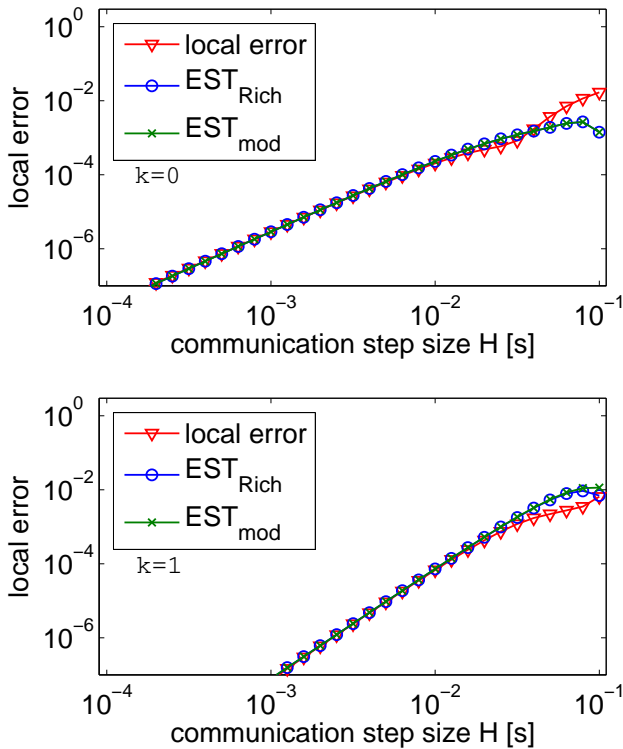


Figure 2: Benchmark Quarter car - co-simulation for a coupled system (2) without feed-through: Local error (“ ∇ ”) and error estimates $\mathbf{EST}_{\text{Rich}}$ (“ \circ ”), $\mathbf{EST}_{\text{mod}}$ (“ \times ”). Constant ($k = 0$, upper plot) and linear extrapolation ($k = 1$, lower plot).

that contains all outputs of the $r \geq 2$ subsystems at $t = T_{n+2}$. The error indicator (8) shows whether the communication step size $H = H_n$ was sufficiently small to meet some user defined error bounds ATOL_j , RTOL_j or not. Analogously to the classical approach [9] a communication step is accepted, if $\text{ERR} \leq 1$. When $\text{ERR} > 1$, then the estimated error was too large and the communication step has to be repeated with a smaller step size to meet the accuracy requirements.

The ratio between the error indicator ERR and its optimal value 1.0 may be used to define a posteriori an “optimal” communication step size

$$H_{\text{opt}} := \alpha H_n \left(\frac{1}{\text{ERR}} \right)^{1/P}$$

with $P = k + 1$ if there exist subsystems with direct feed-through, otherwise $P = k + 2$, a safety factor $\alpha \in [0.8, 0.9]$ to reduce the risk of a rejection of the next communication step if the current step was accepted and k denoting the approximation order of the signal extrapolation for slave inputs $\bar{\mathbf{u}}(t)$. Note, that H_{opt} is always smaller than the current communication step size H_n if the error estimate \mathbf{EST} exceeds the given

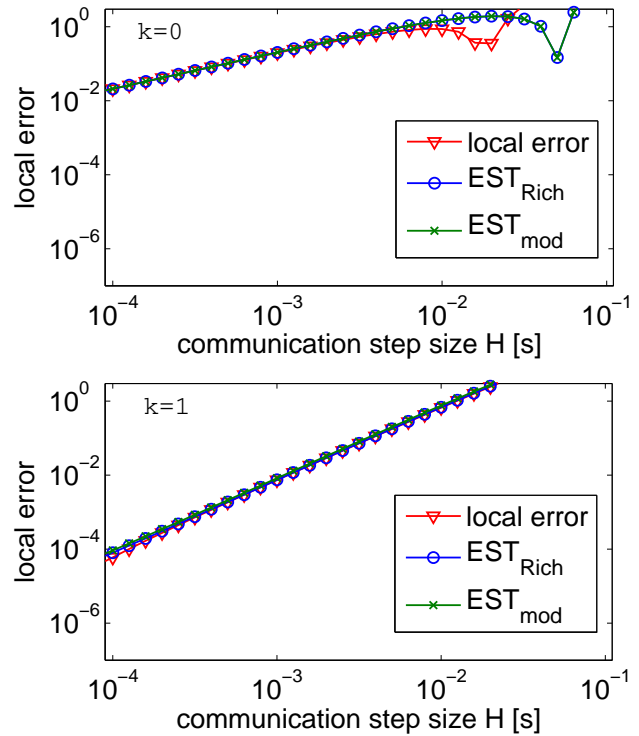


Figure 3: Benchmark Quarter car - co-simulation for a coupled system (2) with feed-through: Local error (“ ∇ ”) and error estimates $\mathbf{EST}_{\text{Rich}}$ (“ \circ ”), $\mathbf{EST}_{\text{mod}}$ (“ \times ”). Constant ($k = 0$, upper plot) and linear extrapolation ($k = 1$, lower plot).

tolerances ($\text{ERR} > 1$).

In practical applications the step size is not allowed to increase nor to decrease too fast [9]. Therefore, parameters $\theta_{\min} \in [0.2, 0.5]$ and $\theta_{\max} \in [1.5, 5]$ are used to restrict the step size change. It is clear that choosing both parameters too small may increase the computational work. Moreover, choosing them too large can increase the risk of rejected steps. The resulting communication step size is given by

$$H_{\text{opt}} = H_n \cdot \min\{\theta_{\max}, \max\{\theta_{\min}, \alpha \cdot (1/\text{ERR})^{1/P}\}\}.$$

4 Generic implementation scheme in FMI 2.0

The Functional Mock-up Interface (FMI) for Model Exchange and Co-Simulation v2.0, see [2, 8], is a standard for the coupling and exchange of models and simulator coupling. A component implementing the FMI is called Functional Mock-up Unit (FMU). It consists of C-functions in source code or preprocessed binaries (like dll or shared object) and an XML description file, that contains all static information

for calling the FMU [4]. The C-functions are called to set and get values in the FMU (FMI-functions `fmiSetReal` and `fmiGetReal`). If the FMU is used for simulator coupling it contains the full model and a simulator (slave). The slave is controlled by a function `fmiDoStep` [13] to process one communication step. With this FMI-function the computation of a communication step in a slave is started with the input parameters `currentCommunicationPoint` (current communication point T_n of the master), `communicationStepSize` (step size H_n of the communication step) and `noSetFMUStatePriorToCurrentPoint`. The Parameter `noSetFMUStatePriorToCurrentPoint` is `true` (`fmiTrue`) if the FMI-function `fmiSetFMUState` will no longer be called for time instants prior to `currentCommunicationPoint`. This is an important information for a slave, that is able to reject communication steps, since the FMU states have to be restored to the last accepted communication point in that case. With the flag `noSetFMUStatePriorToCurrentPoint` the slave can use this information to flush a result buffer. For the Richardson extrapolation based step size control the master has to repeat the simulation from the last accepted communication point, that means it has to save and restore the FMU states to this point for the computation of the reference solution to estimate the error or if a step was rejected because the step size was too large. `fmiGetFMUState` makes a copy of the internal FMU state and returns a pointer to this copy and `fmiSetFMUState` copies the content of the previously copied FMUstate back and uses it as current new FMU state.

A generic implementation scheme for a double step $T_n \rightarrow T_{n+1} \rightarrow T_{n+2}$ with Richardson extrapolation based communication step size control is given by:

- (A) $t_{\text{curr}} = T_n$. Get slave states $slave_{i,n}$, ($i = 1, \dots, r$), calling `fmiGetFMUState` for all r slave FMUs.
- (B) Define subsystem inputs $\bar{\mathbf{u}}_i(t)$ providing derivatives $d^l \bar{\mathbf{u}}_i/dt^l(T_n)$, ($l = 0, \dots, k$) by `fmiSetRealInputDerivatives`.
- (C) Perform large communication step $T_n \rightarrow T_n + 2H = T_{n+2}$ calling `fmiDoStep` for all r slave FMUs and get outputs $\bar{\mathbf{y}}(T_{n+2})$ by `fmiGetRealOutputDerivatives`.
- (D) Reset all slave FMUs to state $slave_{i,n}$, ($i = 1, \dots, r$), calling `fmiSetFMUState`.
- (E) Define subsystem inputs $\bar{\mathbf{u}}_i(t)$ providing derivatives $d^l \bar{\mathbf{u}}_i/dt^l(T_n)$, ($l = 0, \dots, k$) by `fmiSetRealInputDerivatives`.
- (F) Perform first small communication step $T_n \rightarrow T_n + H = T_{n+1}$ calling `fmiDoStep` for all r slave FMUs and get outputs $\bar{\mathbf{y}}(T_{n+1})$ by `fmiGetRealOutputDerivatives`.
- (G) Evaluate $\bar{\mathbf{u}}(T_{n+1}) = \mathbf{c}(\bar{\mathbf{y}}(T_{n+1}))$ and define subsystem inputs $\bar{\mathbf{u}}_i(t)$ providing derivatives $d^l \bar{\mathbf{u}}_i/dt^l(T_{n+1})$, ($l = 0, \dots, k$) by `fmiSetRealInputDerivatives`.
- (H) Perform second small communication step $T_{n+1} \rightarrow T_{n+1} + H = T_{n+2}$ calling `fmiDoStep` for all r slave FMUs and get outputs $\bar{\mathbf{y}}(T_{n+2})$ by `fmiGetRealOutputDerivatives`.
- (I) Evaluate error estimate $\mathbf{EST}_{\text{Rich}}$, error indicator ERR and optimal communication step size H_{opt} .

The function `fmiDoStep` returns `fmiOK` if the communication step was computed successfully until its end. `fmiDiscard` is returned if the slave computed successfully only a subinterval of the communication step, which may occur if the communication step size is too large and the simulation should be repeated with a smaller one. `fmiError` will be returned by `fmiDoStep` if the simulation of the communication step could not be carried out at all.

The capabilities of a slave are described in the XML file. A capability flag `canHandleVariableCommunicationStepSize` set to `true` indicates, that a slave is able to accept variable communication steps, which is important to implement a master with communication step size control. The complete interface description can be found in [13].

5 Numerical test for the FMI compatible master

For developing and testing new master algorithms, a master prototype was implemented by Fraunhofer IIS/EAS and Halle University [4, 6]. This master supports basic functions of FMI for Co-Simulation version 1.0, see [14], and has implemented several sophisticated master algorithms like communication step size control and the rejection of communication steps.

To use the Fraunhofer master with the given FMUs the compiled C-code of the master has to be linked to the slaves binaries, which may even be C-code that

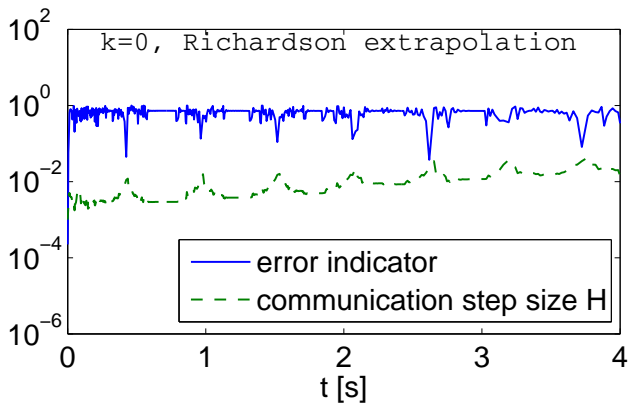


Figure 4: Benchmark Quarter car, displacement-displacement coupling. Error indicator and communication step size of the simulation with Richardson extrapolation based step size control with the Fraunhofer master.

is compiled with the master. The resulting executable consists of the master and the slaves.

For the numerical tests with the Fraunhofer master, the quarter car benchmark with the two slaves “chassis” and “wheel” is implemented in Dymola. We apply the communication step size control strategy from Section 3 with error estimates $\mathbf{EST}_{\text{Rich}}$ and $\mathbf{EST}_{\text{mod}}$ and study the influence of the order k of signal extrapolation and of the coupling strategy (displacement-displacement vs. displacement-force). From a practical viewpoint constant ($k = 0$), linear ($k = 1$) and quadratic ($k = 2$) signal extrapolation are most interesting since higher order extrapolations increase the risk of numerical instability, see also [1]. In all numerical tests, the error tolerances for slave FMUs are chosen two orders of magnitude smaller than the master tolerances ATOL_j , RTOL_j in (8).

Since the current implementation of the Fraunhofer master is limited to constant extrapolation with error estimation by Richardson extrapolation, only these numerical results are depicted in Figure 4 for the displacement-displacement coupling.

Extending these results, we will also consider the numerical simulations in a MATLAB-based test environment for the verification of the theoretical analysis. The numerical test for variable communication step sizes with error estimates $\mathbf{EST}_{\text{Rich}}$ and $\mathbf{EST}_{\text{mod}}$ for constant ($k = 0$) and linear ($k = 1$) extrapolation is depicted for the two coupling strategies in Figures 5 and 6 within the simulation time interval $t \in [0, 4]$. The simulation results for Richardson extrapolation and its modification are identical for $k = 0$, where the small differences in Table 1 are caused by implementation

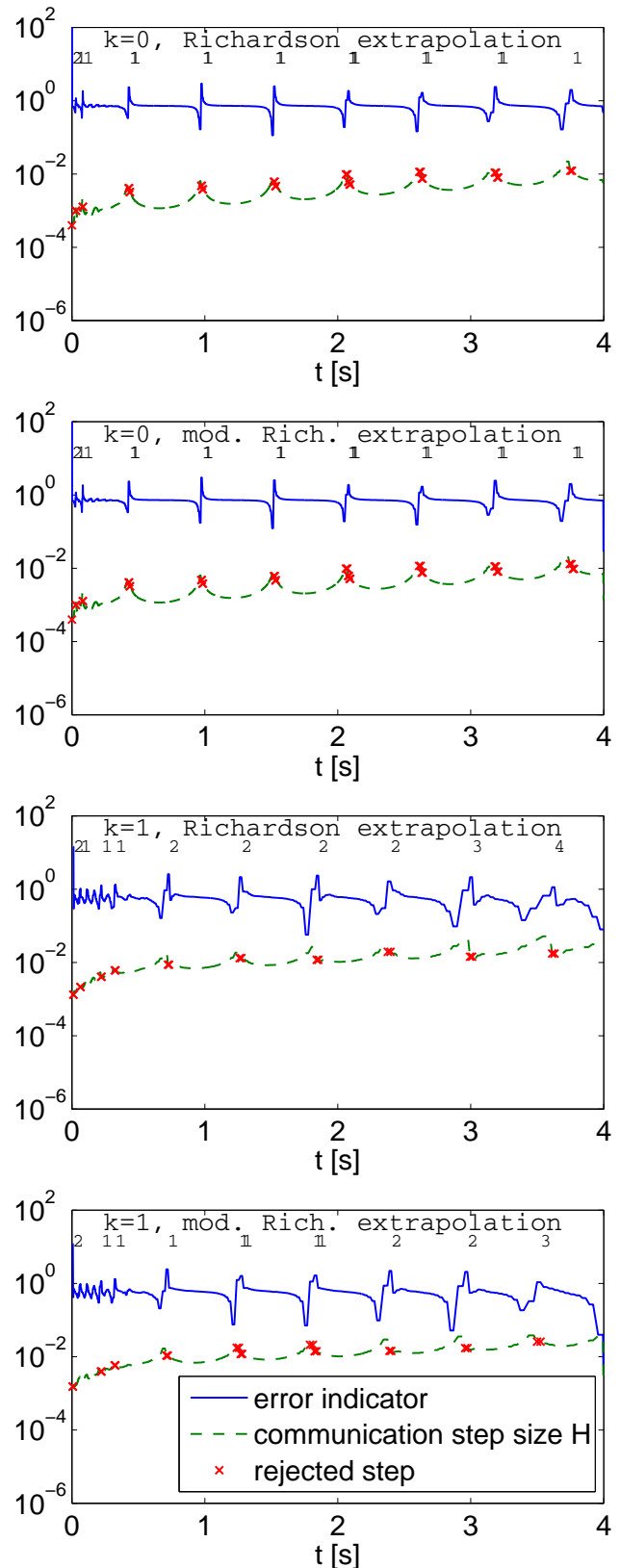


Figure 5: Benchmark Quarter car, displacement-displacement coupling. Number of step rejections, error indicator and communication step size of the simulation with error estimates $\mathbf{EST}_{\text{Rich}}$ and $\mathbf{EST}_{\text{mod}}$.

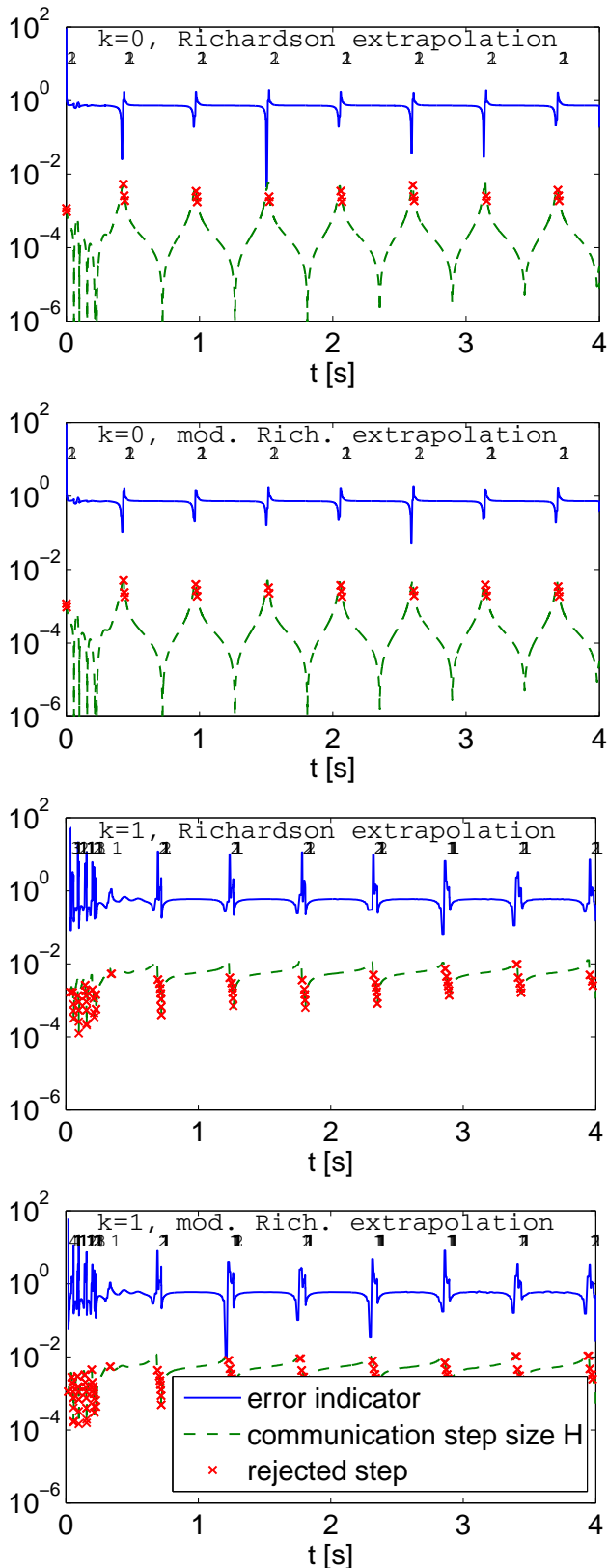


Figure 6: Benchmark Quarter car, displacement-force coupling. Number of step rejections, error indicator and communication step size of the simulation with error estimates EST_{Rich} and EST_{mod} .

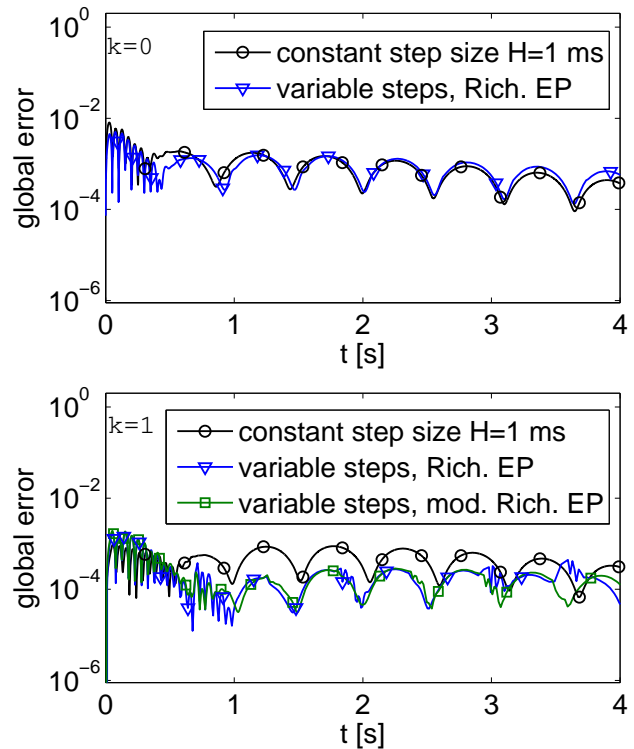


Figure 7: Benchmark Quarter car, displacement-displacement coupling. Global error of the simulation with constant step sizes compared to variable steps based on error estimates EST_{Rich} and EST_{mod} .

issues, since the first (small) Richardson step is saved and therefore the inputs at $t = T_n + H$ have to be interpolated which causes problems if in the big step $T_n \rightarrow T_{n+2}$ only one micro step is taken.

The global error of the numerical solution with step size control is very well controlled to a mean value of 10^{-4} , see Figure 7, which corresponds to the predefined error bounds $ATOL_j = RTOL_j = 10^{-4}$ for displacement-displacement coupling (in the case of displacement-force coupling we use $ATOL_j = RTOL_j = 10^{-3}$). In the transient phase $t \in [0, 0.5]$ very small communication steps are chosen and at later simulation time, the algorithm uses larger communication steps than in the beginning, since the subsystems behave slower and the distance between two communication points for an update of the subsystem inputs is increased. In time intervals, where the larger mass of the chassis has a strong influence on the wheel by changing the direction of motion, this is also triggered by the communication step size control resulting in a reduction of the step size such that the error bounds are met (see slow oscillation of the communication step size and also the rejected steps in Figure 5, where the communication steps are repeated with smaller step

size). In the transient phase, we can see that the error is greater than the pre-defined error bound of 10^{-4} . In this phase smaller steps should be taken, which is triggered correctly by the step size control algorithm. If we compare the simulation results with communication step size control with $\mathbf{EST}_{\text{Rich}}$ and $\mathbf{EST}_{\text{mod}}$ to the simulation with constant communication step size $H = 10^{-3}$ (with a micro tolerance in the subsystems of 10^{-6}) in Figure 7 and Table 1, we can see, that the global error for constant extrapolation is decreasing during the simulation because the step size is not adapted to the solution behaviour. The accuracy is raised, if higher order extrapolations are used. We can also compare the computing times and see that the master algorithm with communication step size control is much faster resulting in a high efficiency (nearly the same mean global error in the simulation time interval compared to constant step sizes with constant extrapolation), even with Richardson extrapolation, where in every communication step the simulation is performed at least twice. Using the modification $\mathbf{EST}_{\text{mod}}$ or a higher order of extrapolation ($k \geq 1$) further improves the simulation results and the computing time. This is a nice example for the advantage of controlling the step size compared to the brute force approach of using always constant communication step sizes.

Furthermore, we observe in Table 1 if we use a micro tolerance of 10^{-8} in the subsystems for the simulation with constant step sizes $H = 10^{-3}$ that the computation time is growing. Moreover, the accuracy of the simulation is improved, since the influence of the discretization error of the solution of the subsystems is reduced. The simulation with step size control with $\text{micTOL}=1\text{e-}6$ on the other hand is robust and reliable and controls the error of the simulation results to the predefined tolerance of $\text{macTOL}=1\text{e-}4$, even if the influence of the discretization error of the subsystems is not neglected.

6 Conclusions

We have discussed the error estimation in co-simulation by classical Richardson extrapolation and by a modified algorithm for a reliable communication step size control based on an extension of the step size control of classical time integration. The local error of the simulation is estimated very well by these strategies.

The communication step size control was applied to a benchmark problem from vehicle dynamics which

Table 1: Benchmark Quarter car - Simulation results for the displacement-displacement coupling.

	k	error	time [s]	steps	rej
$H = 1\text{ms}$, $\text{micTOL}=1\text{e-}6$	0	8.922E-004	33.335	4000	0
	1	4.094E-004	29.295	4000	0
	2	4.138E-004	31.022	4000	0
$H = 1\text{ms}$, $\text{micTOL}=1\text{e-}8$	0	5.301E-004	51.280	4000	0
	1	6.049E-005	43.364	4000	0
	2	1.967E-005	43.073	4000	0
Rich. EP, $\text{micTOL}=1\text{e-}6$, $\text{macTOL}=1\text{e-}4$	0	1.050E-003	28.361	1610	18
	1	3.197E-004	10.379	402	20
	2	3.180E-004	7.050	210	20
Mod. Rich. EP, $\text{micTOL}=1\text{e-}6$, $\text{macTOL}=1\text{e-}4$	0	1.061E-003	19.712	1612	19
	1	3.317E-004	7.057	413	16
	2	2.735E-004	4.835	218	17

was implemented in a master prototype that is compatible to FMI for Co-Simulation v1.0. We have seen that communication step size control is possible, reliable and can improve the performance of the master algorithm significantly, especially the computing time and accuracy.

References

- [1] M. Arnold. Stability of sequential modular time integration methods for coupled multibody system models. *J. Comput. Nonlinear Dynam.*, 5:031003, 2010.
- [2] M. Arnold, T. Blochwitz, C. Clauß, T. Neidhold, T. Schierz, and S. Wolf. FMI-for-CoSimulation. In *The International Journal of Multiphysics. Special Edition: Multiphysics Simulations. Advanced Methods for Industrial Engineering. Selected contributions from 1st Fraunhofer Multiphysics Conference*, pages 345–356, Brentwood, Essex, UK, 2011. Multi-Science Publishing Co. Ltd.
- [3] M. Arnold, C. Clauß, and T. Schierz. Numerical aspects of FMI for Model Exchange and Co-Simulation v2.0. In P. Eberhard and P. Ziegler, editors, *Proc. of The 2nd Joint International Conference on Multibody System Dynamics, Stuttgart, Germany, May 29 - June 1, 2012*, ISBN 978-3-927618-32-9, 2012.
- [4] J. Bastian, C. Clauß, S. Wolf, and P. Schneider. Master for Co-Simulation Using FMI. In C. Clauß, editor, *Modelica Association, Linköping: 8th International Modelica Confer-*

- ence 2011 : Dresden, Germany, 20-22 March 2011, Dresden: Fraunhofer IIS / EAS, 2011.
- [5] M. Busch and B. Schweizer. Numerical stability and accuracy of different co-simulation techniques: Analytical investigations based on a 2-DOF test model. In *Proc. of The 1st Joint International Conference on Multibody System Dynamics, May 25–27, 2010*, Lappeenranta, Finland, 2010.
- [6] C. Clauß, M. Arnold, T. Schierz, and J. Bastian. Master zur Simulatorkopplung via FMI. In X. Liu-Henke, editor, *Tagungsband der ASIM/GI-Fachgruppen STS und GMMS, Wolfenbüttel, 23.02.-24.02.2012*, Ostfalia Hochschule für Angewandte Wissenschaften, Wolfenbüttel, 2012.
- [7] P. Deuffhard and A. Hohmann. *Numerical Analysis in Modern Scientific Computing: An Introduction*. Number 43 in Texts in Applied Mathematics. Springer, 2nd edition, 2003.
- [8] FMI. The functional mockup interface. <http://www.functional-mockup-interface.org/>.
- [9] E. Hairer, S.P. Nørsett, and G. Wanner. *Solving Ordinary Differential Equations. I. Nonstiff Problems*. Springer-Verlag, Berlin Heidelberg New York, 2nd edition, 1993.
- [10] K. Jackson. A survey of parallel numerical methods for initial value problems for ordinary differential equations. *IEEE Transactions on Magnetics*, 27:3792–3797, 1991.
- [11] R. Kübler. *Modulare Modellierung und Simulation mechatronischer Systeme*. Fortschritt-Berichte VDI Reihe 20, Nr. 327. VDI-Verlag GmbH, Düsseldorf, 2000.
- [12] R. Kübler and W. Schiehlen. Two methods of simulator coupling. *Mathematical and Computer Modelling of Dynamical Systems*, 6:93–113, 2000.
- [13] Modelisar. Functional Mock-up Interface for Model Exchange and Co-Simulation v2.0 Beta 3. http://www.functional-mockup-interface.org/specifications/FMI_for_ModelExchange_and_CoSimulation_v2.0_Beta3.pdf, November 2011.
- [14] Modelisar. Functional Mock-up Interface for Co-Simulation. http://www.functional-mockup-interface.org/specifications/FMI_for_CoSimulation_v1.0.pdf, October 2010.
- [15] H. Olsson. Private communication within FMI 2.0 development, June 2011.

On the Formulation of Steady-State Initialization Problems in Object-Oriented Models of Closed Thermo-Hydraulic Systems

Francesco Casella

Dipartimento di Elettronica e Informazione, Politecnico di Milano

Piazza Leonardo da Vinci 32, 20133 Milano, Italy

casella@elet.polimi.it

Abstract

The object-oriented formulation of steady-state initialization for models of closed thermo-hydraulic systems yields singular problems, due to system-wide structural issues. The paper proposes how to solve this problem in an object-oriented fashion, by means of an additional component that helps to uniquely determine the initial conditions of the system. A method based on the analysis of the null space of the Jacobian of the initialization problem and on suitable annotations is also proposed to provide the end user with meaningful, high-level, context-relevant diagnostic messages, in case the singular problem arises. This diagnostic method can also be applied to other cases, such as closed systems with constant density fluid and electrical circuits lacking a ground connection.

Keywords: Thermo-hydraulic system modelling, Initialization, User-friendly error diagnostics

1 Introduction

Dynamic modelling of thermo-hydraulic systems in Modelica is becoming increasingly popular in many application fields: energy conversion systems, air conditioning and ventilation plants for civil and airborne applications, etc.

The steady-state initialization of such models is well-known to be a critical issue. The two typical alternatives available to practitioners are:

- set the initial values of all the state variables to estimates of the steady-state one (the exact value is not known a priori), then simulate a relaxation transient until a steady-state is reached;
- set the initial derivatives of all the state variable to zero and let the tool numerically solve for the exact steady-state values.

The first choice has the advantage that finding a consistent initial state is numerically easy, so the simulation always starts, but the ensuing transient can be problematic, because of potentially large swings of flow variables that can require a very long simulation time and possibly lead to solver failures.

The second choice is more appealing: if the solver converges, the time spent to solve the steady-state initialization problem is typically much lower than the time spent simulating the (meaningless) relaxation transient. However, there are three categories of issues that can prevent getting the desired initial state:

1. the initialization problem is well-posed, but a solution is not found because of convergence failure of the iterative solver (typically, the initial guess values are not close enough to the solution);
2. the initialization problem is structurally well-posed, but the values of some parameters are such that a physically valid solution does not exist;
3. the initialization problem is not structurally well-posed, e.g., it is singular.

Recently proposed improvements [7, 4] based on simplified models and homotopy transformations, have been proved effective to enhance the likelihood of convergence (issue 1.) and also to point out problems stemming from the wrong parametrization of the model (issue 2.). However, they do not address issue 3. at all.

If the thermo-hydraulic system under consideration is closed, i.e., it does not exchange mass with the outside world, initialization problems where all components are set up for steady-state initialization turn out to be singular. The singularity arises at the system level, so it doesn't take place if parts of the closed system are first tested separately, with suitable boundary conditions that make the system an open one, but only

when the final closed system is assembled. This behaviour can be puzzling for inexperienced users.

The goal of this paper is to propose an elegant, object-oriented way to completely determine the initial conditions for closed systems, as well as a method based on numerical analysis and suitable annotations to issue meaningful diagnostics when such system-level singularities do arise, guiding the end-user towards the solution of the problem.

The paper is structured as follows: in Section 2, the structure of steady-state initialization problems in closed circuits is analysed; in Section 3, an object-oriented solution to the problem is proposed and then applied in Section 4 to two case studies. Section 5 discusses how a tool can report the singularity to the end user in a meaningful way, while Section 6 ends the paper with concluding remarks.

2 Singular initialization problems of closed systems

Consider a generic closed thermo-hydraulic system. Dynamic models of such a system contain mass balance equations of three kinds. The first corresponds to mass balances inside control volumes where the stored mass can change over time:

$$\frac{dM_i}{dt} = \sum_j w_{i,j}, \quad i = 1, \dots, N_d, \quad (1)$$

where M_i is the mass contained within the i -th volume, $w_{i,j}$ are the flow variables of its fluid ports, i.e. all the mass flow rates crossing the component boundary through the ports assumed positive when entering, and N_d is the number of control volumes having such dynamic balances. The second kind corresponds to mass balances inside components where the change over time of stored mass is neglected:

$$0 = \sum_j w_{i,j}, \quad i = N_d + 1, \dots, N_d + N_s, \quad (2)$$

where $w_{i,j}$ has the same meaning as above, and N_s is the number of control volumes with such static balances. The third kind corresponds to the mass balance equations formulated over infinitesimally small control volumes spanning each connection set of fluid ports, which are automatically generated by the compiler:

$$0 = \sum_m w_{k,m}, \quad k = 1, \dots, N_c \quad (3)$$

where N_c is the number of connection sets and, for the k -th connection set, $w_{k,m}$ are the flow variables of all the ports belonging to it.

It is essential to note that both in (1) and (2), *all* mass flow rates appearing in the mass balance equations correspond to flow through the ports. Thus, source and sink components, which represent flows exchanged between the system and the outer world, are explicitly excluded. The system is closed, as it cannot exchange mass with the outer environment, but only with other components belonging to it.

Assume now one wants to initialize the system in steady state. In an object-oriented formulation of the system model, which is formed by connecting component models through their ports, one typically selects a steady state initialization option for each component, possibly via some system-level default option which is passed to all the components via inner/outer constructs. This approach is followed by the `Modelica.Fluid` library [6]. This option adds an steady-state initial equation for each control volume with dynamic mass balance:

$$\frac{dM_i}{dt} = 0, \quad i = 1, \dots, N_d. \quad (4)$$

The system of equations (1)-(4) is always singular, because its equations are not linearly independent.

Proof: the sum of all dynamic balance equations (1) minus the sum of all initial equations (4) plus the sum of all static balance equations (2) yields

$$0 = \sum_{i=1 \dots N_d + N_s} w_{i,j}. \quad (5)$$

On the other hand, the sum of all connection equations (3) yields

$$0 = \sum_{k=1 \dots N_c} w_{k,m}. \quad (6)$$

Now, each flow variable in the right-hand-side of (5) belongs to one and only one connection set (for unconnected ports, default connection sets are automatically generated). Therefore each variable appearing in (5) appears once and only once in (6). It follows that the difference between equation (5) and (6), which is a non-trivial linear combination of (1)-(4), gives

$$0 = 0. \quad (7)$$

Hence, equations (1)-(4) are not linearly independent and thus the system is singular, q.e.d.

The physical interpretation of this singularity is that the initial conditions (4) do not give any information on the total amount of mass contained in the system at initialization.

3 Object-oriented formulation of well-posed initialization problems

3.1 Mathematical formulation

One way to make the initialization problem non-singular is to avoid the initial equation (4) for one of the control volumes in the circuit, substituting it with some other equation that makes the problem well posed. This is convenient for those systems where one component has the specific purpose of enforcing the pressure at some point of the circuit.

For example, closed pressurized circuits containing liquid water usually feature dedicated components which determine the pressure level of the system and accommodate the thermal expansion of the fluid: accumulators in domestic heating systems, pressurizers in cooling circuits for PWR nuclear reactors, etc. In this case, the initial conditions for those components are not given as (4), but rather by specifying the initial value of the pressure (and possibly of other variables, depending on the actual level of detail of the model).

Unfortunately, this approach is not always convenient for two reasons. One is that some closed systems that use a compressible fluid as working medium may not contain such a specialized component. The other one, which is more fundamental, is that the extra initial condition which is needed to make the initialization well-posed might refer to the entire system and not just to a specific component. For example, it is often the case that refrigeration circuits must be initialized so that the *total mass* of the fluid contained in the circuit, also known as the *charge*, has a certain value.

In this paper, a modular approach is proposed to solve this issue. The idea is to add an extra component to the system model which contains the equation

$$0 = w_{N_d+N_s+1,1} + w_b, \quad (8)$$

where $w_{N_d+N_s+1,1}$ is the flow variable of the only port of this additional component and w_b is an *unknown* parameter, as well as an extra *initial equation* to completely determine the initial state, such as, e.g.

$$p = p_{start} \quad (9)$$

(where p is the port pressure and p_{start} a known parameter), or, e.g.,

$$\sum_i M_i = M_{start} \quad (10)$$

where M_{start} is again a known parameter.

Including this extra component to the system adds two more equations, e.g., (8),(9) or (8),(10), and two more unknowns, $w_{N_d+N_s+1,1}$ and w_b , to the initialization problem, which thus remains balanced.

If (8) is added to the set of static mass balance equations (2), equation (5) will change to

$$0 = w_b + \sum_{i=1\dots N_s+N_d+1,j} w_{i,j}, \quad (11)$$

and the difference between (11) and (6) will now yield

$$0 = w_b \quad (12)$$

as the term w_b is not a flow through a port, so it is not cancelled out. Therefore, the mass balance equations and initial equations are no longer linearly dependent, and the value of the flow rate w_b potentially entering the system will be computed to be zero during initialization, so the additional component will have no effect on the model during simulation.

It is not trivial to prove that the initialization problem will be non-singular in general. The user building the model should select the extra initial condition (e.g., (9) or (10)) so that the corresponding system of equations uniquely determine the initial state of the system, based on his understanding and expertise.

If a compressible fluid model is employed, typically (9) is a good choice, leading to a numerical problem which is easier to solve than the one obtained by adding (10).

3.2 Modelica formulation

The Modelica code of the initialization component is now presented, based on the mathematical formulation laid out in the previous sub-section, in order to be compatible with the `Modelica.Fluid` library.

```

model ClosedSystemInitializer
  replaceable package Medium =
    Modelica.Media.Interfaces.PartialMedium;
  parameter Medium.AbsolutePressure p_start;
  final parameter Medium.MassFlowRate
    w_b(fixed = false) = 0;
  Modelica.Fluid.Interfaces.FluidPort_a
    port(redeclare package Medium = Medium,
         m_flow(min = 0),
         p(start = p_start));
  Modelica.Blocks.Interfaces.RealInput
    initialConditionResidual;
equation
  0 = port.m_flow + w_b; // Mass balance
  port.h_outflow = 0;
  port.Xi_outflow = zeros(Medium.nXi);
initial equation
  0 = initialConditionResidual;
end ClosedSystemInitializer;

```

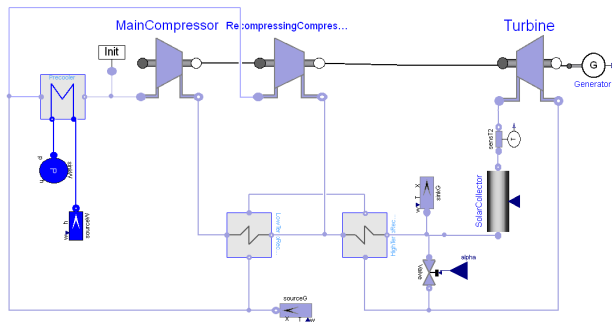


Figure 1: Brayton cycle plant model.

When instantiating the model, the extra initial condition can be specified by connecting a `RealExpression` block containing its residual to the component's input port.

Note that the `min` annotation on the fluid port specifies that the fluid will never flow out of it, so the values of the stream variables `h_outflow` and `Xi_outflow` (which must be given in order to get a balanced model) are never actually used outside the component itself, when computing the incoming stream quantity via the `inStream()` operator [6]. Therefore, once the initial solution of the problem has been found, in particular $w_b = 0$, this additional component has no influence at all on the remaining equations of the closed system.

In order to improve the convergence of the solver when (10) is used as initial condition, it might be possible to use a homotopy transformation, where the simplified initial equation is (9). The initial equation section is correspondingly changed to:

```
initial equation
0 = homotopy(
  actual = initialConditionResidual,
  simplified = port.p - p_start);
```

4 Example applications

4.1 Brayton cycle for power generation

Advanced energy conversion cycles are being considered for high-temperature heat sources, such as central receiver solar plants and IV generation nuclear plant, using supercritical CO_2 as a working medium and a closed Brayton cycle configuration [1, 5], featuring two separate compressors to achieve optimal efficiency of the overall cycle. The object diagram of the plant model is shown in Fig. 1. In order to control the pressure levels in the system, in particular at the main compressor inlet, it is possible to add or remove

mass from two points of the circuit, through appropriate sub-systems which are modelled here as ideally controlled flow rate sources or sinks. When the plant model is considered together with the pressure controller model in a closed-loop configuration, then a full steady-state formulation of the initialization problem, as in (4), is well-posed, because the pressure level (and thus the mass) of the fluid contained in the circuit is determined implicitly by the controller to be equal to the set point.

During the control system design phase, though, it is usually necessary to analyse the dynamic behaviour in open loop around equilibrium points, which in this case means there is no pressure controller and both flow rates are set to zero, so that the system is effectively closed.

In this case, a convenient way to make the initialization problem well-posed is to connect the `ClosedSystemInitializer` component at the compressor inlet, so that it sets the initial pressure at that point. This condition, together with the flow characteristics of the turbo-machines and with the thermal interaction with the heat source and sink, uniquely determines the amount of gas contained in the system and the pressure and temperature distribution.

The model was set up using the `ThermoPowerLight` library, which is a simplified version of the `ThermoPower` library [2, 3] currently under development for optimization studies, and successfully solved using `Dymola 2013`.

4.2 Refrigeration cycle

Refrigeration systems are usually carefully sealed, in order to avoid leaks of refrigerant fluid to the environment, so they always qualify as closed system. The static and dynamic behaviour of the system is heavily influenced by the amount of refrigerant which is initially loaded in the system (the *charge*).

A simple model of a refrigeration circuit, built with the `ThermoPower` library, is shown in Fig. 2. Homogeneous flow and constant heat transfer coefficients are assumed for simplicity; pressure losses along the condenser and evaporator are lumped at the pipe ends.

The initialization problem has been set up by adding an extra condition specifying the total mass contained in the condenser and evaporator pipes. In order to avoid convergence problem, homotopy-based initialization is performed, starting from a simplified model where the condenser inlet pressure is fixed. The model was successfully initialized in steady-state using `Dymola 2013`.

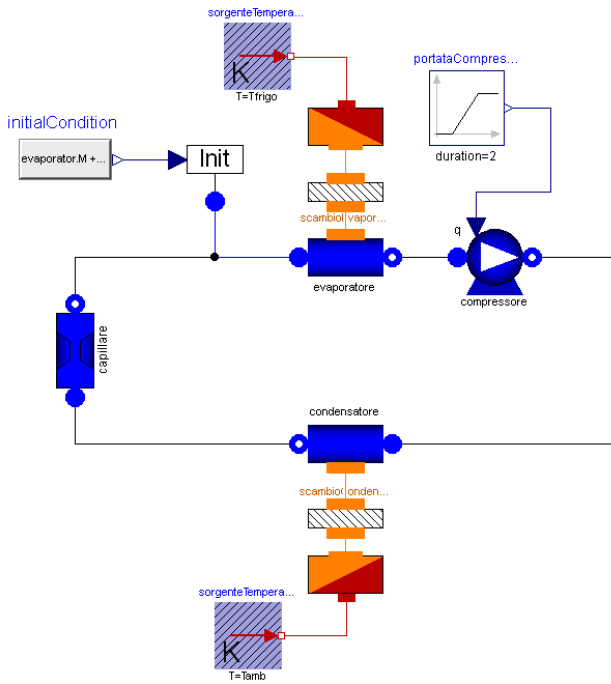


Figure 2: Refrigeration cycle plant model.

5 Diagnostics of ill-posed initialization problems

This section addresses the problem of giving end-users meaningful feedback in case they do not specify the initialization problem correctly and fall into the singular case presented in Sect. 2.

The situation with currently available Modelica tools is not satisfactory. The complete steady-state initialization problem is usually large (hundreds or thousands of unknowns, depending on the degree of detail of the model) and strongly non-linear. The presence of linearly dependent equations makes its Jacobian singular. As a consequence, during iterations the unknowns, which typically include pressures, temperatures and specific enthalpies of control volumes in the model, will fluctuate wildly, causing repeated out-of-bounds errors from the fluid property calculation functions, until the solver eventually gives up. The error messages typically shown to the end user will point to these out-of-bounds errors and, possibly, to the very high condition number of the Jacobian. By no means this diagnostic information points explicitly to the actual root cause, which has been shown in Sect. 2.

One possible solution to this problem is sketched in the remainder of this Section, based on numerical analysis and suitable annotations in the model library.

5.1 Numerical identification of singular subsystems

Let the initialization problem be formulated in residual form (as it is usually the case if nonlinear solvers are to be employed):

$$F(x, \dot{x}, v, p) = 0 \quad (13)$$

where $x \in \mathfrak{R}^n$ is the vector of state variables, $v \in \mathfrak{R}^m$ is the vector of algebraic variables, and $p \in \mathfrak{R}^q$ is the vector of unknown parameters (having the attribute `fixed = false`). The analysis of under- and over-constrained initialization problems, though extremely interesting, is outside the scope of this paper, so it is assumed here that the initialization problem is square, i.e., the function returning the residuals of the dynamic and initial equations of the system is $F : \mathfrak{R}^n \times \mathfrak{R}^n \times \mathfrak{R}^m \times \mathfrak{R}^q \rightarrow \mathfrak{R}^{2n+m+q}$. Define the vectors of unknowns

$$z = \begin{bmatrix} x \\ \dot{x} \\ v \\ p \end{bmatrix}, \quad (14)$$

$z \in \mathfrak{R}^{2n+m+q}$, and let F_z be the Jacobian matrix of the function F with respect to z . The ill-posed initialization problem described in Section 2 leads to a singular Jacobian, since it is shown there that there exist a non-zero vector v such that

$$F_z v = 0 \quad (15)$$

having non-zero entries (plus/minus one) only in correspondence to the mass balance equations and initial equations. If many disconnected closed systems exist in the model, then there will be a corresponding number of linearly independent vectors v_j that satisfy eq. (15). Note that there might also be other such vectors because of other parts of the problem being singular on their own. The set of linearly independent v_j 's satisfying eq. (15) spans the so-called nullspace or kernel of F_z .

It is now possible to identify the set(s) of linearly dependent equations in the initialization problem by numerically computing the nullspace of the Jacobian F_z , i.e. a set of orthonormal vectors v_j that forms a basis for the nullspace. For each of these vectors, all entries whose absolute value is greater than a suitable small threshold identify the equations in the initial problem that are part of a singular subsystem, which can be reported to the end-user.

The selection of the threshold might be critical, because due to numerical approximations, the zero entries will actually have a small non-zero value, and it might not be trivial to avoid false positives or false negatives. It is then essential to use a state-of-the-art numerically robust algorithm to compute the orthonormal basis of the nullspace of F_z , with the lowest possible effect of numerical rounding errors on the result.

Since the sum of square of the elements of each basis vector is one, it is expected that there will be a sharp difference between the elements that correspond to the involved equations (which will have order of magnitude of one) and the other ones (which will have order of magnitude of machine ε , around 10^{-16} for double precision arithmetic). Some experimentation in real-life-sized test cases is however necessary to fine tune such a method, but this has not been possible yet for the lack of available implementation of the method in Modelica compilers.

Another issue to be taken care of is the proper handling of cases when $N > 1$ disconnected closed sub-systems are present in the model. In this case, the basis of the nullspace will contain N orthonormal vectors, but there is no guarantee that the non-zero entries of one of them will only refer to one sub-system. The reason of this fact is that the orthonormal basis of the nullspace is not unique, as other perfectly valid bases can be found by taking one and multiplying its vectors by an orthogonal matrix, i.e., obtaining a basis which is rotated with respect to the previous one. It is then possible that the basis vectors which are obtained from the SVD decomposition are linear combinations of the ones that each refer to one singular sub-system.

However, once that these basis vectors v_1, \dots, v_N have been found for the null space, obtaining vectors whose non-zero entries point to one sub-system is fairly straightforward, by means of a pivoting algorithm. The idea is to look for linear combinations of the originally found vectors that have the least possible number of non-zero entries. A possible sketch of the algorithm is:

1. Build the matrix $M = [m_{i,j}] = [v_1 \ v_2 \ \dots \ v_N]$.
2. Look for the rows with more than one element having absolute value greater than the threshold; if none are found, stop.
3. If at least row one is found, select among them the elements $m_{i,j}$ and $m_{i,k}$ which have the largest absolute value, such that $|m_{i,j}| \geq |m_{i,k}|$.

4. Subtract column j multiplied by $m_{i,k}/m_{i,j}$ from column k .
5. Go to step 2.

When the algorithm terminates, the elements above threshold of each column of M correspond to the singular equations of just one sub-system.

Last, but not least, it is essential to point out that the proposed analysis must be performed on the *full initialization problem* (13), prior to any optimization such as alias elimination, symbolic simplification, and BLT partitioning of the problem. Such optimizations make it easier and more efficient to solve the problem from a numerical point of view, but effectively destroy the structural information that is required to issue meaningful diagnostic errors to the end user.

For example, equations (4) could be used to statically determine that the initial values of the mass derivatives are zero, so these equations could be eliminated from the set of initial equations and all instances of $\frac{dM_i}{dt}$ could be replaced by zeros in equation (1). Obviously, this makes it impossible to point out to the root cause of the problem, which lies in the equations (4). It is of course possible to first try solving the problem with all optimizations active, and only if that fails, generate the Jacobian F_z of the full problem and evaluate it using the values of the unknowns thus found.

As a final remark, although these methods can be fairly expensive in terms of CPU load, the computation is only performed once after the initialization has failed, and a waiting time of a few seconds (or even of a minute or two) is largely preferable to quickly getting diagnostic output that gives no meaningful information in order to trace the root cause of the problem.

5.2 High-level error diagnostics for the user

It is possible to infuse in the model additional expert knowledge from the modeller, which can further help the end user to identify the root cause of the singularity. In the case under consideration, thanks to the analysis carried out in Section 2, the expert library developer knows that a bad choice of steady-state initial equations will lead to a singular problem, in which those equations will form a singular sub-system. It would then be possible to annotate those equations with meaningful error messages, that will be reported to the end-user in case they are found to be part of such a singular sub-system, e.g.:

```
initial equation
der(M) = 0
```

```

annotation(PartOfSingularSystemError =
  "Ill-posed initial conditions for closed
  system.\n Please connect a
  ClosedSystemInitializer component to
  the system to completely specify the
  initial state");

```

Instead of just reporting the raw set of equations which form a singular subsystem, the tool could report their associated error annotation strings, that would help even an inexperienced user to fix the problem quickly. A hot link to the documentation of the `ClosedSystemInitializer`, e.g. marked-up using a `modelica:// URL`, could lead with one click to a documentation page that explains the problem in more detail, and possibly even link to this paper on the Web for further information. Note that annotations on equations are allowed by the Modelica language grammar, though they have never been used so far for any definite purpose.

5.3 Application to other cases

The mechanism proposed in this Section could also be used for diagnostic purposes in other situations that suffer of the same syndrome, namely, a sub-set of the system equations being singular due to the system-wide structural issues.

A first example is models of thermo-hydraulic systems with constant density fluid, which is a modelling assumptions sometimes used if the working medium is liquid and one is not interested in the very fast pressure dynamics, nor in the thermal expansion and buoyancy effects. In this case, the sets of equations (1) and (4) are empty, and all the mass balance equations of the control volumes that make up the system are contained in set (2). The linear combination of equations mentioned in Section 2 still yields $0 = 0$, so the system is singular. In this case, however, the singularity does not involve the initial equations, so both the initialization problem and the simulation problem are singular. An annotation could then be applied to the static mass balances of the components using incompressible fluid, e.g. with error message "Closed system with constant density fluid yields singular system of equations. Please add a component such as pressurizer or accumulator to determine the pressure in the circuit uniquely".

Another example is the well-known case of electrical circuits with missing ground component. In this case, the system of dynamic equations is singular because the pin voltages are not uniquely determined. Various solutions have been proposed, based on structural analysis, to issue meaningful diagnos-

tic messages to the end user. Using the numerical method proposed in this paper, it would suffice to add the `PartOfSingularSystemError` annotation to the equation $v = p.v - n.v$ contained in the `OnePort` partial model, e.g. with error message "Electrical circuit without ground connection yields singular system of equations. Please connect a ground component to the circuit where appropriate".

6 Conclusions

Models of closed thermo-hydraulic circuits can lead to singular steady-state initialization problems due their system-wide structure. This paper proposes a way to solve this problem in an object-oriented fashion, by means of an additional component that helps to uniquely determine the initial conditions of the system.

A method based on the analysis of the null space of the Jacobian of the initialization problem is also proposed to provide the end user with meaningful, high-level, context-relevant diagnostics, by suitably annotating those equations that might potentially lead to such singular problems. This allows the library designer to infuse expert knowledge about potential system-level issues, helping inexperienced end-users to pin-point the root cause of the problem easily, contrary to the current state-of-the-art with existing tools, that will output hard-to-understand low-level numerical diagnostic messages.

The proposed method for high-level singular system diagnostics can also be applied to other cases, such as closed circuits with constant-density fluid, or electrical circuits lacking a ground connection.

References

- [1] F. Casella and P. Colonna. Development of a Modelica dynamic model of solar supercritical CO₂ Brayton cycle power plants for control studies. In *Proceedings of the Supercritical CO₂ Power Cycle Symposium*, pages 1–7, Boulder, Colorado, USA, May 24–25 2011.
- [2] F. Casella and A. Leva. Modelica open library for power plant simulation: Design and experimental validation. In P. Fritzson, editor, *Proceedings 3rd International Modelica Conference*, pages 41–50, Linköping, Sweden, Nov. 3–4 2003. Modelica Association. <http://www.elet.polimi.it/upload/casella/thermopower/>.

- [3] F. Casella and A. Leva. Modelling of thermo-hydraulic power generation processes using Modelica. *Mathematical and Computer Modeling of Dynamical Systems*, 12(1):19–33, Feb. 2006.
- [4] F. Casella, M. Sielemann, and L. Savoldelli. Steady-state initialization of object-oriented thermo-fluid models by homotopy methods. In C. Clauss, editor, *Proceedings 8th International Modelica Conference*, pages 86–96, Dresden, Germany, Mar. 20–22 2011. Modelica Association.
- [5] V. Dostal. *A Supercritical Carbon Dioxide Cycle for Next Generation Nuclear Reactors*. PhD thesis, MIT, 2004.
- [6] R. Franke, F. Casella, M. Sielemann, K. Proelss, M. Otter, and M. Wetter. Standardization of thermo-fluid modeling in Modelica.Fluid. In F. Casella, editor, *Proceedings 7th International Modelica Conference*, pages 122–131, Como, Italy, Sep. 20–22 2009. The Modelica Association.
- [7] M. Sielemann, F. Casella, M. Otter, C. Clauß, J. Eborn, S. E. Mattsson, and H. Olsson. Robust initialization of differential-algebraic equations using homotopy. In C. Clauss, editor, *Proceedings 8th International Modelica Conference*, pages 75–85, Dresden, Germany, Mar. 20–22 2011. Modelica Association.

Probability-One Homotopy for Robust Initialization of Differential-Algebraic Equations

Michael Sielemann

Deutsches Zentrum für Luft- und Raumfahrt, Robotics and Mechatronics Center,
System Dynamics and Control, Münchner Strasse 20, 82234 Wessling, Germany

Abstract

An evolution of the recently introduced operator homotopy (\cdot) is proposed, which further improves the solution of difficult initialization problems. The background and motivation for this approach are discussed and it is demonstrated how to apply it for electrical and fluid systems. The key difference to the earlier approach is the supporting theory, which guarantees that the method converges globally with probability one.

Keywords: Initialization, DAE, homotopy, nonlinear equations

1 Introduction

A dynamic model describes how the state variables and thus the entire system behave over time. The state variables define the current condition of the model and have to be initialized when simulation starts. For this purpose, Modelica provides language constructs to define initial conditions such as initial equation sections [12]. The resulting constraints and all equations and algorithms that are utilized during the simulation form the initialization problem. Based on its solution, all variables, derivatives and pre-variables are assigned consistent values before the simulation starts.

Mathematically, the resulting problem is an initial value problem for a differential algebraic equation system (DAE) with $\dim(\mathbf{f}) = nx + nw$ equations:

$$\mathbf{f}(\dot{\mathbf{x}}, \mathbf{x}, \mathbf{w}, t) = \mathbf{0}, \mathbf{x}(t) \in \mathbb{R}^{nx}, \mathbf{w}(t) \in \mathbb{R}^{nw}, t \in \mathbb{R}$$

Here, \mathbf{x} is the vector of state variables and \mathbf{w} is the vector of algebraic unknowns. For simplicity of the discussion, we assume that the DAE has no hybrid part and is index-reduced, i.e. it has index 1, which means that the following expression is regular:

$$\begin{bmatrix} \frac{\partial \mathbf{f}}{\partial \dot{\mathbf{x}}} & \frac{\partial \mathbf{f}}{\partial \mathbf{w}} \end{bmatrix}$$

Note that all the following results still hold for hybrid, higher index DAEs with small adaptations. Initialization means to provide consistent initial values for $\dot{\mathbf{x}}_0$, \mathbf{x}_0 , \mathbf{w}_0 so that the DAE is fulfilled at the initial time t_0 . Since these are $2 \cdot nx + nw$ unknowns and the DAE has $nx + nw$ unknowns, additional nx equations must be provided which are called "initial equations" in Modelica:

$$\mathbf{g}(\dot{\mathbf{x}}_0, \mathbf{x}_0, \mathbf{w}_0, t_0) = \mathbf{0}, \dim(\mathbf{g}) = nx$$

The most often used initial equations are:

$$\mathbf{g}(\dot{\mathbf{x}}_0, \mathbf{x}_0, \mathbf{w}_0, t_0) = \dot{\mathbf{x}}_0 = \mathbf{0}$$

that is, steady-state initialization.

The result is usually a nonlinear system of algebraic equations, which has to be solved numerically. This does not always work right away for industrial problems as the commonly employed gradient-based local algorithms [2, 10, 3], such as the damped Newton method, provide local convergence only (even when using globalizations such as trust regions).

Modelica allows users to describe any model mathematically, which makes it highly flexible and powerful for simulation of heterogeneous multi-domain physical systems. However, this also means that no knowledge of the mathematical character of the problem equations can be introduced into the solver. Instead, an algorithm has to work on a general numerical problem (in contrast to domain-specific algorithms for nonlinear problems).

As a result, the success to solve initialization problems of state-of-the-art implementations of Modelica tools depends on the choice of iteration variables and the guess values for these variables defined with the start attribute. As a result it may become difficult for a library developer to provide a robust initialization capability.

Since a model becomes useless whenever initialization fails, and the current state-of-the-art is not fully

satisfactory in this regard, we conclude that more reliable and robust methods are needed for a wider application of the Modelica modeling language by practitioners.

In a previous paper, we introduced a homotopy operator in [19]. It maps $\text{homotopy}(\text{actual} = \dots, \text{simplified} = \dots)$ to $\lambda \cdot \text{actual} + (1 - \lambda) \cdot \text{simplified}$. Successful application examples were given for electronic circuits and multibody systems in [19] and for power plants in [1].

This homotopy operator is typically used to simplify governing equations of components, sweep boundary conditions and the like. The advantages of this approach are that the concept is simple and easy to understand. Also, it was successfully tested on relevant test cases. It has certain limitations however, in particular that the homotopy map is hard-wired into the language specification, that convergence is based on heuristics, and that a naive application can lead to singular problems (e.g., with a singular Jacobian at $\lambda = 0$).

The objective of this contribution is thus to propose a more powerful homotopy operator, which can be used as the original one, enables a declarative definition of arbitrary homotopy maps, and allows global convergence via probability-one homotopy, an established method from topology.

2 Theory

2.1 Definitions

We first define a generic nonlinear algebraic problem with a vector of unknowns $\mathbf{z} = [\mathbf{x}_0; \mathbf{x}_0; \mathbf{w}_0]$ and residuals $\mathbf{F} = [\mathbf{f}; \mathbf{g}]$.

Then, a homotopy is a continuous deformation from one map to another via the homotopy map $\rho(\mathbf{z}, \lambda)$. The homotopy map is a map with one higher dimension as it additionally depends on λ , the homotopy or continuation parameter. The corresponding under-determined system of equations $\rho(\mathbf{z}, \lambda) = \mathbf{0}$ can be followed using continuation algorithms.

Homotopy maps are carefully constructed such that for one value of the homotopy parameter, e.g., $\lambda = 0$, the equation system is easy to solve and for another value, e.g., $\lambda = 1$, the equation system is the one of interest, i.e., $\mathbf{F}(\mathbf{z}) = \mathbf{0}$.

Then, the root finding procedure works as follows. A curve $(\mathbf{z}, 0)$ is followed from $\rho(\mathbf{z}, 0) = \mathbf{0}$ along $\rho(\mathbf{z}, \lambda) = \mathbf{0}$ until $\lambda = 1$ as $\rho(\mathbf{z}, 1) = \mathbf{F}(\mathbf{z})$. This curve $\{\mathbf{z} | \rho(\mathbf{z}, \lambda) = \mathbf{0}\}$ is called the root curve $\rho^{-1}(\mathbf{0})$.

Root curves have to be followed numerically and

therefore they must not contain singularities such as bifurcations or divergence to $\pm\infty$. Also, they must not be closed loops without crossing $\lambda = 1$ (so called isolaes).

2.2 Probability-one homotopy

It is possible to prove that a problem satisfies these requirements using a particular type of homotopy method called probability-one homotopy. This method allows to avoid running into one of the ill-posed traces and thus delivers global convergence. It requires posing the problem equations \mathbf{F} and ρ in a particular fashion and was used with vast success in domain-specific simulation to resolve the convergence issues motivating this paper, in particular in analog electronic circuit simulation [14, 24, 16].

Informally, the key elements of probability-one homotopy are

- A well-defined random element to guarantee the full rank of the Jacobian matrix of ρ ,
- A boundedness argument, and
- An embedding, which essentially corresponds to the simplifications of component governing equations applied in [19, 1].

In order to summarize the supporting theory, transversality to zero [26] is defined.

Definition 1. Let $U \subset \mathbb{R}^m$ and $V \subset \mathbb{R}^n$ be open sets, and let $\rho : U \times [0, 1] \times V \rightarrow \mathbb{R}^n$ be a C^2 map. ρ is said to be transversal to zero if the Jacobian matrix $\partial\rho$ has full rank on $\rho^{-1}(\mathbf{0})$.

Here, $n = nx + nw$ holds. In the definition, an additional parameter dependency on a random vector $\mathbf{a} \in \mathbb{R}^n$ is shown. This is the random element mentioned above. The Jacobian matrix of ρ is $\partial\rho$. It is a $n \times (2n + 1)$ matrix and can be written as concatenation of three sub-matrices.

$$\partial\rho = \begin{bmatrix} \frac{\partial\rho}{\partial\mathbf{a}} & \frac{\partial\rho}{\partial\lambda} & \frac{\partial\rho}{\partial\mathbf{z}} \end{bmatrix} \quad (1)$$

Similarly to $\rho^{-1}(\mathbf{0})$ introduced above, we can now consider $\rho_{\mathbf{a}}^{-1}(\mathbf{0})$ as a set of root curves. Formally, we define it as follows.

$$\rho_{\mathbf{a}}^{-1}(\mathbf{0}) = \{(\mathbf{a}, \lambda, \mathbf{z}) | \mathbf{a} \in \mathbb{R}^n, \\ 0 \leq \lambda < 1, \\ \mathbf{z} \in \mathbb{R}^n, \\ \rho(\mathbf{a}, \lambda, \mathbf{z}) = \mathbf{0}\}$$

The following theorem, which is based on differential geometry and the Parametrized Sard's Theorem, is a generic formulation of probability-one homotopy methods.

Theorem 1. *Let $\mathbf{F} : \mathbb{R}^n \rightarrow \mathbb{R}^n$ be a C^2 map, $\rho : \mathbb{R}^n \times [0, 1] \times \mathbb{R}^n \rightarrow \mathbb{R}^n$ a C^2 map, and $\rho_{\mathbf{a}}(\lambda, \mathbf{z}) = \rho(\mathbf{a}, \lambda, \mathbf{z})$. Suppose that*

1. ρ is transversal to zero, and, for each fixed $\mathbf{a} \in \mathbb{R}^n$,
2. $\rho_{\mathbf{a}}(0, \mathbf{z})$ has a unique nonsingular solution \mathbf{z}_0 ,
3. $\rho_{\mathbf{a}}(1, \mathbf{z}) = \mathbf{F}(\mathbf{z})$.

Then, for almost all $\mathbf{a} \in \mathbb{R}^n$, there exists a zero curve Γ_a of $\rho_{\mathbf{a}}$ emanating from $(0, \mathbf{z}_0)$, along which the Jacobian matrix $\partial \rho_{\mathbf{a}}$ has full rank. If, in addition,

4. $\rho_{\mathbf{a}}^{-1}(\mathbf{0})$ is bounded, then Γ_a reaches a point $(1, \mathbf{z}^*)$ such that $\mathbf{F}(\mathbf{z}^*) = \mathbf{0}$. Furthermore, if $\partial \mathbf{F}(\mathbf{z}^*)$ has full rank, then Γ_a has finite arc length.

This theorem is due to Watson [26] and is therefore called Watson's Theorem in this work. In order to apply this theorem, homotopy maps are constructed to meet prerequisites (2) and (3) by design. Prerequisite (1) may be trivial to verify for some homotopy maps and harder for others, in which λ and \mathbf{a} are involved nonlinearly. According to [27], prerequisite 4 may be hard to verify and often is a "deep result" as (1)–(4) holding implies the *existence* of a solution to $\mathbf{F}(\mathbf{z}) = \mathbf{0}$.

A remark is in order on the statement of probability one. This characteristic of the theorem is inherited from the Parametrized Sard's Theorem and is motivated by probability of failure being 0 in the sense of a Lebesgue measure. Figuratively speaking, this means that the set of points leading to failure forms at most an $n - 1$ dimensional manifold inside n -dimensional space, that is, it does not occupy any "volume".

Informally, Watson's Theorem can be understood as a statement on the probability of singularities along a continuation path. A bifurcation for instance may occur on a problem fulfilling this theorem. But a random variation of the parameter vector \mathbf{a} will be sufficient to avoid the singularity on a following attempt (with probability one). On problems with more than one solution, this choice of \mathbf{a} determines what solution the homotopy map converges to.

A number of additional theorems on probability-one homotopies are reviewed in [18]; herein, the given one shall suffice.

3 Implementation in Modelica Tools

3.1 Convergence proofs

For applications, the key issue is to show how a problem satisfies the given theorem. While some applications successfully utilize general physical principles such as conservation of energy (see [23] for instance), research by the authors [18] shows that it is not possible to generalize such proofs to arbitrary physical domains. This was mentioned in reference [19] already. Instead, problem-specific arguments are used in this contribution. They are introduced together with homotopy maps below.

Conceptually, they work via a general no-gain property, as exposed by, e.g., electric resistors, diodes, and transistors, and via saturation (an amplifier for instance has a constant gain only until the amplified signal reaches the supply voltage).

3.2 Declarative definition of arbitrary homotopy maps

Modelica is meant to allow a declarative problem description. That is one in which no information has to be provided on *how* to solve the problem. Instead, the problem itself is described. The solution algorithms are encapsulated in the language compilers and simulators.

In order to be useful for practitioners, the notion of problem-specific homotopy maps has to be integrated into simulation tools. The goal was thus to extend the declarative description to homotopies.

Using Modelica, one structures a model in terms of classes and objects. Therefore, it is proposed to specify a homotopy map $\rho_{\mathbf{a}}(\mathbf{z}, \lambda)$ on the level of the equation set of the model classes, too.

In order to implement the suggested approach, it is proposed to utilize a built-in operator, `lambda()`¹. Any expression involving λ , which describes the problem-specific homotopy map, can be written using the operator `lambda()`. It is used for each occurrence of variable λ . This operator may return a value in $[0, 1]$ during the numeric solution of algebraic equation systems and strictly 1 during the generation of simulation results. If the operator `lambda()` is used without an argument then a *single-phase homotopy map* is implemented. If integer arguments are used then a

¹Note that the previously proposed operator can be expressed using this operator. Furthermore, except for the cases with multi-stage homotopies, the previously proposed homotopy-operator can be used since `lambda()=homotopy(simplified=0, actual=1)`.

homotopy map is implemented, which consists of n phases, where n is the maximum over all arguments of the operator `lambda()`. For example, when using `lambda(1)` and `lambda(2)`, then a homotopy map is implemented in which λ_1 values are first swept from 0 to 1. After this is finished, λ_2 values are swept from 0 to 1. λ has to be swept from 0 to 1 during these sequential continuation runs of λ_i in order to infuse the random element required by theory².

In order to simulate a given model efficiently, Modelica tools may apply symbolic preprocessing steps. A step that has to be considered in the context of homotopy is equation sorting. A typical example of a sorting algorithm used for equation-based, object-oriented modeling languages is the Block Lower Triangular (BLT) transformation [6], using a graph-theoretical algorithm by Tarjan [21]. Conceptually speaking, the continuation must be applied to the equation set as a whole. That is, all the equations that are either directly or indirectly influenced by the homotopy operator have to be solved simultaneously.

Note that if any of the probability-one homotopy theorems, such as the one introduced before, is fulfilled, then a large fraction of potential problems is avoided. For example, no singular Jacobian matrix at $\lambda = 0$ can arise.

3.3 Test implementation

In order to validate the methodology, a test implementation was developed. It was based on the Modelica compiler Dymola[®] in versions 7.3 and 6.1. This test implementation utilized the LOCA continuation algorithms of Trilinos [8] and had the following properties.

- It provided three options for the treatment of the suggested homotopy operator. Normally, it was expanded according to a homotopy map. Alternatively, reduced equation sets were obtained by inlining the homotopy expression assuming $\lambda_i = 1.0$ or $\lambda_i = 0.0$. In the latter case, maximum structural simplification of the equation system resulted.
- The user was able to manually prescribe whether to use homotopy initialization or not. This was

²When using the homotopy operator with integer arguments, several distinct continuation runs have to be started as the trajectories will in general not be smooth at the joining point of traces in any λ_i and λ_{i+1} . In general, the trajectories will be continuous but not differentiable. Even if a continuation algorithm manages to “hop over” such a joining point, starting continuation separately may be more efficient.

an important feature for library development and debugging, and may be useful for users, too.

- Verbose information on the homotopy was optionally provided, which was useful for library development and debugging. In particular, the homotopy traces were visualized. Like this, it was possible to reconstruct what happened during the solution of the simplified problems and the homotopy transformation.

Several implementation aspects such as automatic scaling and solver configuration via XML files have been described in [20, 18] and equally apply to this solver implementation.

4 Application Examples

As mentioned in the introduction, the use of probability-one homotopy is particularly well-developed in the area of analog electronic circuit simulation [23, 24, 25, 14, 13, 9, 7, 22, 11, 17, 29, 16]. First application examples are thus based on this work.

4.1 Operational amplifier $\mu A741$

The first example is an operational amplifier, which was discussed in [19] already. It uses bipolar junction transistors. Results are presented on probability-one homotopy using two different homotopy maps, the variable stimulus and the variable gain method. They are introduced next.

4.1.1 Variable Stimulus

Melville et al. [14] proposed the Variable Stimulus Probability-One Homotopy. Its homotopy map is as follows.

$$\rho(\mathbf{z}, \lambda) = (1 - \lambda)G(\mathbf{z} - \mathbf{a}) + \mathbf{F}(\mathbf{z}, \lambda) \quad (2)$$

Here, the residual equations $\mathbf{F}(\mathbf{z}, \lambda)$ are posed in the nodal analysis form [4] and the node voltages of the nonlinear elements are multiplied by λ . Therefore, the influence of the nonlinear elements is removed from the circuit at $\lambda = 0.0$ and a linear circuit has to be solved. The matrix G defines the leakage from voltage sources of value \mathbf{a} . These voltage sources and the associated vector \mathbf{a} provide the random element needed in the probability-one approach. The leakage matrix G is a diagonal matrix with coefficients G_{leak} .

In order to substantiate that the Variable Stimulus Homotopy is globally convergent, Melville et al. [14] utilize Watson's Theorem as stated in section 2.2. Their arguments are as follows.

- The homotopy map (2) is twice continuously differentiable if and only if the device models used to assemble the residual equations in nodal form $\mathbf{F}(\mathbf{z})$ are sufficiently smooth. It is assumed that this is fulfilled.
- The homotopy map ρ is transversal to zero as $\partial\rho/\partial\mathbf{a}$ in (1) is a diagonal matrix with entries $-(1-\lambda)\cdot G_{\text{leak}}$. For $\lambda < 1$, this matrix has full rank.
- $\rho_{\mathbf{a}}(\mathbf{z}, 0)$ has a unique non-singular solution, because for $\lambda = 0$ the circuit consists of resistors and voltage sources only. Such a linear problem has a unique non-singular solution.
- $\rho_{\mathbf{a}}(\mathbf{z}, 1) = \mathbf{F}(\mathbf{z})$ because the leakage circuitry is removed completely at $\lambda = 1$ and each nonlinear device model is stimulated by the actual voltage.
- The zero set $\rho_{\mathbf{a}}^{-1}(\mathbf{0})$ is bounded due to the no-gain property of the actual circuit and any partially stimulated circuit with leakage circuitry.

Additionally, Melville et al. [14] make the engineering assumption that the Jacobian of $\rho_{\mathbf{a}}$ has full rank at the solution \mathbf{z}^* .

This Variable Stimulus Homotopy can be implemented on analog circuits using the proposed homotopy operator. First, a model of a NPN bipolar junction transistor is provided (see listing 1).

Here, three functions `iCollectorNpn()`, `iEmitterNpn()`, and `iBaseNpn()` are used to establish the collector, emitter, and base currents respectively. In order to implement the leakage circuitry, a model instance of a class is attached to each connection set (see listing 2).

Note the negative sign in front of the summation of the currents of the pins. This is necessary as the nodal analysis form [4] summarizes the currents going into the components attached to a node.

According to the experiments of Melville et al. [14], the solution trajectories of this homotopy are "much smoother" than those of the generic homotopy maps mentioned in section 2.1 of [19]. Additionally, "the action is spread out evenly over all values of λ ".

4.1.2 Variable Gain

Melville et al. [14] also proposed the Variable Gain homotopy, which is similar to the Variable Stimulus homotopy but addresses bipolar transistors differently. Instead of multiplying the terminal voltages of all nonlinear elements by λ , the forward current gain α_F and the reverse current gain α_R are multiplied by λ . The simplified problem with $\alpha_F = 0$ and $\alpha_R = 0$ therefore consists of resistors, voltage sources, and diodes only.

$$\rho(\mathbf{z}, \lambda) = (1 - \lambda)G(\mathbf{z} - \mathbf{a}) + \mathbf{F}(\mathbf{z}, \lambda\alpha) \quad (3)$$

Again, the residual equations $\mathbf{F}(\mathbf{z}, \lambda\alpha)$ are posed in the nodal analysis form [4]. Due to the diodes, the leakage circuitry is not necessary to avoid floating nodes. However, it is still included in this homotopy to provide the random element to avoid bifurcations [14].

Originally, the Variable Gain homotopy was implemented as a two-stage procedure. First, the Variable Stimulus homotopy was used to solve the $\lambda = 0$ problem of the Variable Gain homotopy. Then, continuation was started on the Variable Gain homotopy map (3) and the actual problem was solved. Today, Variable Gain Homotopy is commonly understood as what was originally labeled the "hybrid approach" in reference [14]. A local gradient-based algorithm is used to solve the $\lambda = 0$ problem and the continuation is applied directly on the Variable Gain homotopy map. The robust convergence of a local gradient-based algorithm on the $\lambda = 0$ problem is justified by Melville et al. [14] in case of norm-reducing algorithms (algorithms using so-called globalizations) by the work of Duffin [5]. The single-stage procedure is "two to three times faster than using homotopy alone" [14].

In order to show that the Variable Gain Homotopy is globally convergent, Melville et al. [14] again utilize Watson's Theorem. Their arguments are as follows.

- As before, the homotopy map (3) is twice continuously differentiable if and only if the device models used to assemble the residual equations in nodal form $\mathbf{F}(\mathbf{z})$ are sufficiently smooth. Again, it is assumed that this is fulfilled.
- The homotopy map ρ is transversal to zero as $\partial\rho/\partial\mathbf{a}$ in (1) is a diagonal matrix with entries $-(1-\lambda)\cdot G_{\text{leak}}$. For $\lambda < 1$, this matrix has full rank.
- $\rho_{\mathbf{a}}(\mathbf{z}, 0)$ has a unique non-singular solution, because for $\lambda = 0$ the circuit consists of resistors, voltage sources, and diodes only. Duffin [5] proved that such a problem has a unique solution.

```

1 model NPN
2 // Connectors
3 Modelica.Electrical.Analog.Interfaces.Pin C "Collector";
4 Modelica.Electrical.Analog.Interfaces.Pin B "Base";
5 Modelica.Electrical.Analog.Interfaces.Pin E "Emitter";
6
7 // Parameters
8 parameter Real af = 0.995 "Forward current gain";
9 parameter Real ar = 0.5 "Reverse current gain";
10
11 equation
12 C.i = iCollectorNpn(
13     lambda()*B.v, lambda()*C.v, lambda()*E.v, af, ar);
14 E.i = iEmitterNpn(
15     lambda()*B.v, lambda()*C.v, lambda()*E.v, af, ar);
16 B.i = iBaseNpn(
17     lambda()*B.v, lambda()*C.v, lambda()*E.v, af, ar);
18 end NPN;

```

Listing 1: NPN transistor model using variable stimulus

- $\rho_a(\mathbf{z}, 1) = \mathbf{F}(\mathbf{z})$ because the leakage circuitry is removed completely at $\lambda = 1$ and each nonlinear device model uses the nominal forward and reverse current gains.
- The zero set $\rho_a^{-1}(\mathbf{0})$ is bounded as Melville et al. [14] showed. This is due to the results of [28], which showed that bipolar transistors exhibit the no-gain property as long as the absolute values of the current gains remain less than or equal to one.

This Variable Gain Homotopy can be implemented on analog circuits using the proposed homotopy operator. Again, a model of a NPN bipolar junction transistors is given (see listing 3).

As before, three functions `iCollectorNpn()`, `iEmitterNpn()`, and `iBaseNpn()` are used to establish the collector, emitter, and base currents respectively. Instead of the terminal voltages, the current gains are multiplied with λ . The leakage circuitry can be implemented using model instances of the class listed in section 4.1.1 and is not repeated here.

According to Melville et al. [14], this is their fastest converging homotopy map. In particular, “the time required to solve a system of operating point equations with this homotopy [map] is not more than two to three times slower than the time required to solve the same equations by less widely convergent methods”.

4.1.3 Results

Figure 1 shows robustness profiles [20] of both homotopy-based solvers and a local gradient-based algorithm [15] for comparison. A robustness profile, in essence, shows the probability of convergence over the quality of a start iterate. The probability of convergence P_{conv} is estimated by sampling. The quality of the start iterate in turn is measured by a scaled distance of the start iterate $\tilde{\mathbf{z}}$ to the next solution $\tilde{\mathbf{s}}_j$. Algorithms, which deliver constantly full probability of convergence, i.e., $P_{conv} = 1$, independently of the quality of the start iterate, are called globally convergent herein.

4.2 Inverter Chain

This example involves Metal-Oxide-Semiconductor Field-Effect Transistors (MOSFETs). The circuit it is based on is not found in practical devices, however, it can be scaled via the number of inverters in the chain. For the results discussed here, $n = 50$ inverters were used. For MOSFETs, the ATANSH homotopy is currently state of the art.

4.2.1 Arc-Tangent Shichman-Hodges

The Arc-Tangent Shichman-Hodges or ATANSH model was proposed by Roychowdhury and Melville [16, 17] for probability-one homotopy and large-scale integrated circuits of metal-oxide

```

1 model ElectricalNode
2 // Connectors
3 parameter Integer n=0 "Number of pins"
4 annotation(Evaluate=true, Dialog(connectorSizing=true));
5 Modelica.Electrical.Analog.Interfaces.Pin pin[n] "Pin array";
6
7 // Parameters
8 parameter Real Gleak "Leakage";
9 parameter Real a "Random source voltage";
10
11 equation
12 0 = -sum(pin[:].i) + (1.0 - lambda())*Gleak*(pin[1].v-a);
13 for i in 1:n-1 loop
14   pin[i].v = pin[i+1].v;
15 end for;
16 end ElectricalNode;

```

Listing 2: Electrical node class

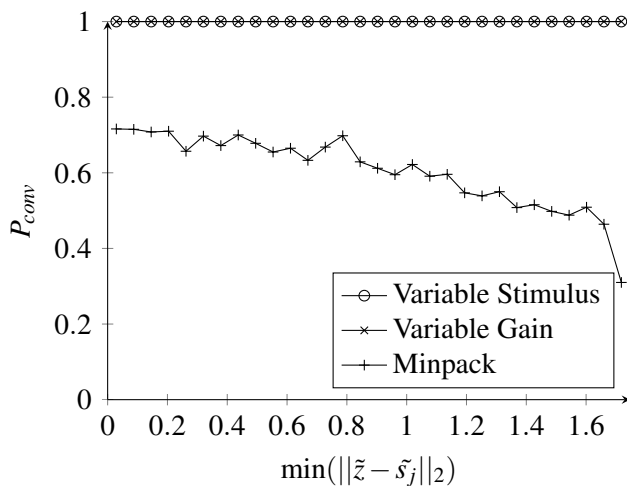


Figure 1: Robustness profiles [20] for Operational Amplifier 741 (60/60/1000 samples per bin)

semiconductor field-effect transistors. Conceptually, it is similar to the Variable Gain homotopy in that it varies key nonlinearity in component models. The ATANSH model uses two homotopy parameters λ_1 and λ_2 . Parameter λ_1 influences the drain–source driving point characteristic without affecting the gain. Parameter λ_2 in turn controls the transfer characteristic, i.e., the gain, without affecting the driving point characteristic.

$$\rho(\mathbf{z}, \lambda, \lambda_1, \lambda_2) = (1 - \lambda)G(\mathbf{z} - \mathbf{a}) + F(\mathbf{z}, \lambda_1, \lambda_2) \quad (4)$$

The ATANSH MOS homotopy model is a single-piece model. The drain–source current I_{ds} is given via

the following equation [16].

$$I_{ds} = \frac{\beta}{2} [V'_{gs}(V_{gb}, V_{db}, V_{sb}, \lambda_2, \lambda_1)]^2 \cdot h(V_{db} - V_{sb}, \lambda_1) \quad (5)$$

Roychowdhury and Melville [16, 17] remark that their probability-one homotopy map is a heuristic. In an attempt to justify its success, Watson's Theorem as stated in section 2.2 is considered.

- The homotopy map (4) is twice continuously differentiable if and only if the device models used to assemble the residual equations in nodal form $\mathbf{F}(\mathbf{z})$ are sufficiently smooth. For the given MOS model this is fulfilled.
- The homotopy map ρ is transversal to zero as $\partial\rho/\partial\mathbf{a}$ in (1) is a diagonal matrix with entries $-(1 - \lambda) \cdot \text{Gleak}$. For $\lambda < 1$, this matrix has full rank.
- $\rho_{\mathbf{a}}(\mathbf{z}, 0)$ has a unique non-singular solution, because for $\lambda = 0$ the circuit consists of resistors, voltage sources, and simplified MOS transistors only. At $\lambda_1 = 0$ and $\lambda_2 = 0$ the simplified MOS devices become two-terminal almost-linear resistors. It is a reasonable engineering assumption to assume that such a problem has a unique non-singular solution.
- $\rho_{\mathbf{a}}(\mathbf{z}, 1) = \mathbf{F}(\mathbf{z})$ because the leakage circuitry is removed completely and each MOS device model is restored to its original form.

```

1 model NPN
2 // Connectors
3 Modelica.Electrical.Analog.Interfaces.Pin C "Collector";
4 Modelica.Electrical.Analog.Interfaces.Pin B "Base";
5 Modelica.Electrical.Analog.Interfaces.Pin E "Emitter";
6
7 // Parameters
8 parameter Real af = 0.995 "Forward current gain";
9 parameter Real ar = 0.5 "Reverse current gain";
10
11 equation
12 C.i = iCollectorNpn(
13     B.v, C.v, E.v, lambda()*af, lambda()*ar);
14 E.i = iEmitterNpn(
15     B.v, C.v, E.v, lambda()*af, lambda()*ar);
16 B.i = iBaseNpn(
17     B.v, C.v, E.v, lambda()*af, lambda()*ar);
18 end NPN;

```

Listing 3: NPN transistor model using variable gain

- The zero set $\rho_a^{-1}(\mathbf{0})$ is bounded due to the no-gain property of the actual circuit and the simplified one with leakage circuitry and simplified MOS device models.

Additionally, one can make the engineering assumption that the Jacobian of ρ_a has full rank at the solution \mathbf{z}^* .

This MOS model for probability-one homotopy can be implemented using the proposed homotopy operator. Listing 4 illustrates this on an n-channel MOS transistor.

Function `idsNchannel()` implements equation (5) for this type of transistor. Note how the `lambda()` operator is used as described in section 3.2 with an integer argument. As Roychowdhury and Melville [16] first ramp λ_2 and then λ_1 , their homotopy is implemented using $\lambda_2 = \text{lambda}(1)$ and $\lambda_1 = \text{lambda}(2)$. The leakage circuitry can be implemented using model instances of the class listed in section 4.1 and is not repeated here.

Roychowdhury and Melville [16, 17] report that local gradient-based algorithms are two to three times faster than the ATANSH homotopy on average *if they converge*. They additionally provide data to show however that the ATANSH homotopy took “considerably less time to obtain the DC operating point of the circuit than conventional methods took to give up” on their test cases. This illustrates that the extra wall time is an acceptable price to pay for robust convergence on large-scale problems.

4.2.2 Results

See figure 2 for results on using probability-one homotopy methods³ and on using local gradient-based algorithms in comparison.

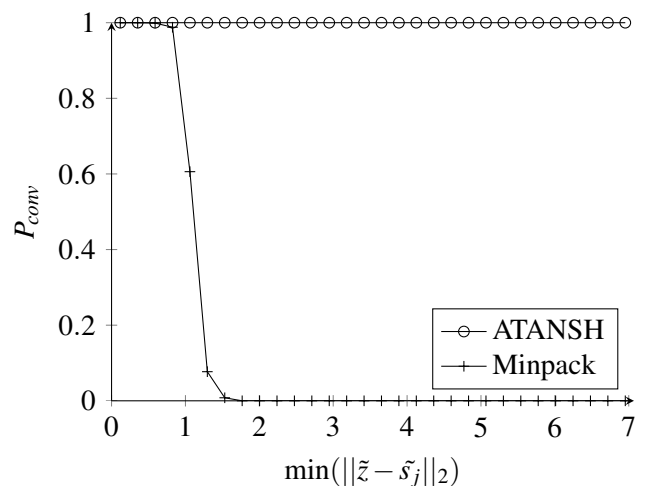


Figure 2: Robustness profiles [20] for Inverter Chain (60/1000 samples per bin)

³The ATANSH homotopy map cannot be compared to the variable gain or variable stimulus homotopy maps. The reason is that they are specific to a type of transistor, either the MOSFET or the BJT.

```

1 model NMOS
2 // Connectors
3 Modelica.Electrical.Analog.Interfaces.Pin G "Gate";
4 Modelica.Electrical.Analog.Interfaces.Pin D "Drain";
5 Modelica.Electrical.Analog.Interfaces.Pin S "Source";
6 Modelica.Electrical.Analog.Interfaces.Pin B "Bulk";
7
8 equation
9 // Drain-source current according to ATANSH
10 D.i = idsNchannel(G.v-B.v, D.v-B.v, S.v-B.v,
11     lambda(1), lambda(2));
12 S.i = -D.i;
13 // Gate, source
14 G.i = 0;
15 B.i = 0;
16 end NMOS;

```

Listing 4: MOS-FET model using ATANSH

4.3 Air distribution network

In this section, a basic but robust probability-one homotopy for thermo-fluid dynamic applications with unidirectional flow is introduced and applied to an Air Distribution test case. This is a thermo-hydraulic example with pipes transporting gases under wall friction and heat transfer, heat loads in cabin volumes, fans and so on. More details are given in [18].

4.3.1 Unidirectional Thermofluid Probability-One Homotopy

The notion of a nodal approach for probability-one homotopy is adopted. Therefore, the mass and energy balances are addressed in this context. Pressure is a potential variable and thus the established approach of leakage circuitry used in sections 4.1.1, 4.1.2, and 4.2.1 can be applied trivially. Therefore, the components implementing the mass balance in the homotopy map are written in nodal form as follows.

$$\rho_{\text{hyd}}(\mathbf{z}_{\text{hyd}}, \mathbf{z}_{\text{th}}, \lambda) = (1 - \lambda) G_{\text{hyd}}(\mathbf{a}_{\text{hyd}} - \mathbf{z}_{\text{hyd}}) + \mathbf{F}_{\text{hyd}}(\mathbf{z}_{\text{hyd}}, \mathbf{z}_{\text{th}}, \lambda) \quad (6)$$

The subscript in $\rho_{\text{hyd}}(\mathbf{z}_{\text{hyd}}, \mathbf{z}_{\text{th}}, \lambda)$ refers to the mass balance as hydraulic part. Consequently, $\mathbf{z}_{\text{hyd}} = \mathbf{p}$, i.e., the vector of unknowns of this part of the homotopy map is the vector of unknown pressures. G_{hyd} is the hydraulic leakage, \mathbf{a}_{hyd} is the vector of pressure values introducing the random element required by probability-one homotopy. The vector of residual equations $\mathbf{F}_{\text{hyd}}(\mathbf{z}_{\text{hyd}}, \mathbf{z}_{\text{th}}, \lambda)$ for the hydraulic part are

the mass balances, that is the sums of the connection set mass flow rates. Of course these residual equations also depend on \mathbf{z}_{th} , the vector of thermal unknowns. These can be either temperatures or specific enthalpies. As it does only matter to the model of thermodynamic properties which one is used and all equations can be transformed accordingly, it is assumed without loss of generality that they correspond to temperature, i.e., $\mathbf{z}_{\text{th}} = \mathbf{T}$.

For the thermal part the situation is more involved. As the temperatures or specific enthalpies z_{th} are not potentials (note that their values are not equal over all connectors in a connection set in the general case), a mechanistic application of the concept to the energy balance will fail. A modified nodal homotopy map component for the energy balance is written in the dimension of a specific enthalpy. It can equally be used with and without conduction (analogous to the leakage in the hydraulic part).

$$\rho_{\text{th}}(\mathbf{z}_{\text{hyd}}, \mathbf{z}_{\text{th}}, \lambda) = (1 - \lambda) G_{th}(\mathbf{a}_{th} - \mathbf{z}_{th}) + \mathbf{F}_{th}(\mathbf{z}_{\text{hyd}}, \mathbf{z}_{th}, \mathbf{a}_{th}, \lambda) \quad (7)$$

Here, the thermal node value \mathbf{z}_{th} is used in the specific enthalpy computation. The mass flow rate leaving the connection set is the sum of the mass flow rates over the outlet connectors plus the mass flow rate due to leakage in equation (6). The superscripts \pm on the mass flow rates indicate that they have been limited to a positive or negative epsilon flow using a C^2 regularization.

The residual equations involving λ are as follows.

$$\mathbf{F}_{\text{th}}(\mathbf{z}_{\text{hyd}}, \mathbf{z}_{\text{th}}, \mathbf{a}_{\text{th}}, \lambda) = (1 - \lambda) h_{pT}(\mathbf{a}_{\text{hyd}}, \mathbf{a}_{\text{th}}) \quad (8)$$

$$+ \lambda \frac{\sum_{\text{inlets}} \dot{m}_i^+ \cdot h_i}{\sum_{\text{inlets}} \dot{m}_i^+}$$

$$- h_{pT}(\mathbf{z}_{\text{hyd}}, \mathbf{z}_{\text{th}})$$

The homotopy map has been established in terms of the connection set equations. Optionally, one may create embeddings in the device models. For wall friction correlations, a convex combination of a secant approximation through some operating point and the actual wall friction correlation was successfully tested. Heat transfer may be established equally based on secant approximations or even zero heat transfer at $\lambda = 0$.

In order to substantiate that the thermo-fluid homotopy is globally convergent, theorem 1 (Watson's Theorem) is applied. The arguments are as follows.

- The homotopy map based on components (6) and (7) is twice continuously differentiable if and only if the device models used to assemble the residual equations in nodal form $\mathbf{F}(\mathbf{z})$ are sufficiently smooth. It is assumed that this is fulfilled.
- The homotopy map ρ is transversal to zero as $\partial\rho/\partial\mathbf{a}$ with $\mathbf{a} = [\mathbf{a}_{\text{hyd}}; \mathbf{a}_{\text{th}}]$ in (1) contains a diagonal matrix with entries $-(1 - \lambda) \cdot G$ with $G = [G_{\text{hyd}}; G_{\text{th}}]$ if a conductance is used. If the conductance is not used, i.e., $G_{\text{th}} = 0$, then $\partial\rho/\partial\mathbf{a}$ contains $-(1 - \lambda) \cdot G_{\text{hyd}}$ for the hydraulic part. For the thermal part, $\partial\rho/\partial\mathbf{a}$ contains $(1 - \lambda) c_p$. In any case $\partial\rho/\partial\mathbf{a}$ and the Jacobian (1) have full rank for $\lambda < 1$.
- The homotopy map $\rho_{\mathbf{a}}(\mathbf{z}, 0)$ has a unique non-singular solution, because for $\lambda = 0$ the circuit consists of adiabatic linear pressure loss models and boundary conditions only. Such a problem has a unique non-singular solution.
- $\rho_{\mathbf{a}}(\mathbf{z}, 1) = \mathbf{F}(\mathbf{z})$ because the balance equations are restored completely at $\lambda = 1$ and each device model exposes the actual behavior.
- For the hydraulic part, the zero set $\rho_{\mathbf{a}}^{-1}(\mathbf{0})$ is bounded due to the no-gain property of the pressure loss correlations. See [18] for further details. For the thermal part, the zero set is bounded due to the Second Law of Thermodynamics⁴.

⁴At a first glance, one could argue that going from $\lambda = 0$ to $\lambda = 1$ is not necessarily "forward" in time. However, the Second Law is used here on a set of steady-state problems. Therefore, no issues arise from the "direction" of time.

The code for a model class to be instantiated in each connection set is given in listing 5. This node model implements the homotopy map on the thermodynamic balance equations of mass and energy, in particular, equation (6) in lines 30 and 31 and equation (7) in lines 42 to 47. The implementation of the device models is straight-forward. As an example, in listing 6, the steady-state part of a simple dynamic pipe model is presented (the transient equations do not matter for initialization and are thus omitted for readability).

4.3.2 Results

Figure 3 shows a robustness profile for the resulting unidirectional thermo-fluid dynamics probability-one homotopy. The results illustrate that the proposed homotopy map and the probability-one homotopy method provide robust convergence, even in light of large variations of the start iterate and random vector.

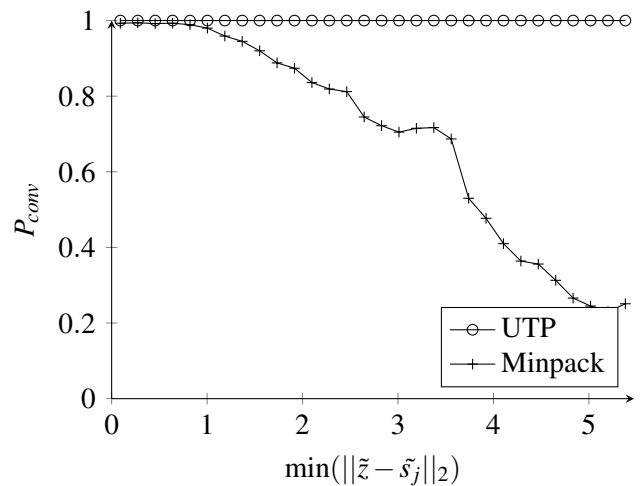


Figure 3: Robustness profiles for Air Distribution (60/1000 samples per bin)

5 Conclusions

The key result is that the theoretically predicted global convergence of probability-one homotopy can be realized in practice. This can be inferred from figures 1 to 3.

The associated coercivity proofs and the construction of underlying embeddings are rather involved however and require considerable understanding and a substantial investment in engineering time.

References

- [1] F. Casella, L. Savoldelli, and M. Sielemann. Steady-state initialization of object-oriented thermo-fluid models by homotopy methods. In *Proceedings of Eighth International Modelica Conference*, Dresden, Germany, March 2011.
- [2] J. E. Dennis and R. B. Schnabel. *Numerical methods for unconstrained optimization and nonlinear equations*. SIAM Classics in Applied Mathematics, 1996.
- [3] P. Deuffhard. *Newton Methods for Nonlinear Problems. Affine Invariance and Adaptive Algorithms*. Springer Verlag, 2004.
- [4] P. Dimo. *Nodal analysis of power systems*. Taylor & Francis, 1975.
- [5] R. Duffin. Nonlinear networks IIa. *B. Am. Math. Soc.*, 53:963–971, 1947.
- [6] H. Elmqvist. *A Structured Model Language for Large Continuous Systems*. PhD thesis, Lund University, Department of Automatic Control, Sweden, May 1978.
- [7] M. Green and R. Melville. Sufficient conditions for finding multiple operating points of dc circuits using continuation methods. In *IEEE International Symposium on Circuits and Systems*, pages 117–120, Seattle, 1995.
- [8] M. A. Heroux, R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps, A. G. Salinger, H. K. Thornquist, R. S. Tuminaro, J. M. Willenbring, A. Williams, and K. S. Stanley. An overview of the Trilinos project. *Acm. T. Math. Software.*, 31(3):397–423, 2005.
- [9] Y. Inoue. A practical algorithm for DC operating-point analysis of large-scale circuits. *Electronics and Communications in Japan (Part III: Fundamental Electronic Science)*, 77(10):49–62, 1994.
- [10] C. T. Kelley. *Solving nonlinear equations with Newton's method*. SIAM Classics in Applied Mathematics, 2003.
- [11] W. Mathis, L. Trajkovic, M. Koch, and U. Feldmann. Parameter embedding methods for finding DC operating points of transistor circuits. In *Third international specialist workshop on Nonlinear Dynamics of Electronic Systems, NDES 1995*, pages 147–150, Dublin, Ireland, July 1995.
- [12] S. Mattsson, H. Elmqvist, M. Otter, and H. Olsson. Initialization of hybrid differential-algebraic equations in Modelica 2.0. In *Proceedings of the Second International Modelica Conference*, 2002.
- [13] R. Melville, S. Moinian, P. Feldmann, and L. Watson. Sframe: An efficient system for detailed DC simulation of bipolar analog integrated circuits using continuation methods. *Analog. Integr. Circ. S.*, 3(3):163–180, 1993.
- [14] R. C. Melville, L. Trajkovic, S.-C. Fang, and L. T. Watson. Artificial parameter homotopy methods for the DC operating point problem. *IEEE T. Comput. Aid. D.*, 12(6):861–877, June 1993.
- [15] J. J. Moré, B. S. Garbow, and K. E. Hillstrom. User guide for MINPACK-1. Technical Report ANL-80-74, Argonne National Laboratory, 1980.
- [16] J. Roychowdhury and R. Melville. Delivering global DC convergence for large mixed-signal circuits via homotopy/continuation methods. *IEEE T. Comput. Aid. D.*, 25(1):66–78, January 2006.
- [17] J. S. Roychowdhury and R. C. Melville. Homotopy techniques for obtaining a DC solution of large-scale mos circuits. In *Proceedings of the 33rd Design Automation Conference*, pages 286–291, 1996.
- [18] M. Sielemann. *Device-Oriented Modeling and Simulation in Aircraft Energy Systems Design*. PhD thesis, Technical University of Hamburg-Harburg, Institute of Thermo-Fluid Dynamics, 2012.
- [19] M. Sielemann, F. Casella, M. Otter, C. Clauss, J. Eborn, S. Mattsson, and H. Olsson. Robust initialization of differential-algebraic equations using homotopy. In *Proceedings of Eighth International Modelica Conference*, Dresden, Germany, March 2011.
- [20] M. Sielemann and G. Schmitz. A quantitative metric for robustness of nonlinear algebraic equation solvers. *Math. Comput. Simulat.*, 81(12):2673–2687, 2011.

- [21] R. Tarjan. Depth-first search and linear graph algorithms. *SIAM J. Comput.*, 1:146–160, 1972.
- [22] L. Trajkovic and W. Mathis. Parameter embedding methods for finding DC operating points: formulation and implementation. In *1995 International Symposium on Nonlinear Theory and its Applications, NOLTA 1995*, pages 1159–1164, Las Vegas NE, USA, December 1995.
- [23] L. Trajkovic, R. Melville, and S.-C. Fang. Passivity and no-gain properties establish global convergence of a homotopy method for DC operating points. In *IEEE International Symposium on Circuits and Systems*, volume 2, pages 914–917, May 1990.
- [24] L. Trajkovic, R. C. Melville, and S.-C. Fang. Finding DC operating points of transistor circuits using homotopy methods. In *Proc. IEEE Int Circuits and Systems Symposium*, pages 758–761, 1991.
- [25] L. Trajkovic, R. C. Melville, and S.-C. Fang. Improving DC convergence in a circuit simulator using a homotopy method. In *Proc. Custom Integrated Circuits Conf. the IEEE 1991*, 1991.
- [26] L. T. Watson. Globally convergent homotopy methods: A tutorial. *Appl. Math. Comput.*, 31:369–396, May 1989.
- [27] L. T. Watson. Probability-one homotopies in computational science. *J. Comput. Appl. Math.*, 140:785–807, 2002.
- [28] A. N. Willson Jr. The no-gain property for networks containing three-terminal elements. *IEEE T. Circuits. Syst.*, 22(8):678–687, August 1975.
- [29] K. Yamamura, T. Sekiguchi, and Y. Inoue. A fixed-point homotopy method for solving modified nodal equations. *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on*, 46(6):654–665, 1999.

```

1 model ThermoFluidDynamicsNode
2   replaceable package Medium = PartialPureSubstanceMedium;
3
4   // Connectors
5   parameter Integer nInlets = 0 "Number of inlets"
6     annotation(Evaluate=true, Dialog(connectorSizing=true));
7   parameter Integer nOutlets = 0 "Number of outlets"
8     annotation(Evaluate=true, Dialog(connectorSizing=true));
9   Modelica.Fluid.Interfaces.FluidPort_a inlet[nInlets](
10    redeclare package Medium = Medium);
11   Modelica.Fluid.Interfaces.FluidPort_b outlet[nOutlets](
12    redeclare package Medium = Medium);
13
14   // Parameters
15   parameter Medium.AbsolutePressure a_hyd "Random pressure";
16   parameter Medium.Temperature a_th "Random temperature";
17   parameter Real G_hyd "Leakage in hydraulic part"
18
19   // Variables
20   Medium.AbsolutePressure p "Pressure in node";
21   SI.MassFlowRate m_flow_plus[nInlets] "Limited inlet flow";
22 equation
23   // Hydraulic part
24   for i in 1:nInlets loop
25     inlet[i].p = p;
26   end for;
27   for i in 1:nOutlets loop
28     outlet[i].p = p;
29   end for;
30   0 = (1-lambda())*G_hyd*(a_hyd - p) +
31     sum(inlet[:].m_flow) + sum(outlet[:].m_flow);
32
33   // Thermal part, no conductance
34   for i in 1:nInlets loop
35     // Hypothetical case
36     inlet[i].h_outflow = Medium.h_default;
37   end for;
38   for i in 1:nOutlets loop
39     // Actual case
40     outlet[i].h_outflow = Medium.h_pT(p, T);
41   end for;
42   0 = ((1-lambda())*Medium.h_pT(a_hyd, a_th) +
43     lambda() * sum({
44       m_flow_plus[i]*
45       inStream(inlet[i].h_outflow) for i in 1:nInlets}
46     )/sum({m_flow_plus[i] for i in 1:nInlets}) -
47     Medium.h_pT(p, T));
48   m_flow_plus[:] = f(inlet[:].m_flow, ...);
49 end ThermoFluidDynamicsNode;

```

Listing 5: Thermo-fluid dynamics node class

```

1 model Pipe
2   replaceable package Medium = PartialPureSubstanceMedium;
3
4   // Connectors
5   Modelica.Fluid.Interfaces.FluidPort_a port_a[nInlets](
6     redeclare package Medium = Medium);
7   Modelica.Fluid.Interfaces.FluidPort_b port_b[nOutlets](
8     redeclare package Medium = Medium);
9
10  // Parameters
11  parameter SI.Length diameter "Pipe inside diameter";
12  parameter SI.Length length "Pipe length";
13  parameter SI.Length Delta "Surface roughness";
14  final parameter SI.Area heatTransferArea =
15    Modelica.Constants.pi*diameter*length;
16  parameter SI.Temperature T_amb "Ambient temperature";
17  parameter SI.Pressure dp_nominal "Nominal dp";
18
19  // Variables
20  SI.SpecificEnthalpy dh "Change of h over device"
21  SI.CoefficientOfHeatTransfer kc;
22  Real effectiveness "NTU effectiveness";
23  SI.Density rho "Upstream density";
24  SI.DynamicViscosity eta "Upstream dynamic viscosity";
25  SI.SpecificHeatCapacity cp "At constant pressure";
26  SI.ThermalConductivity lambda "Thermal conductivity";
27
28  equation
29    // Static mass balance
30    port_a.m_flow + port_b.m_flow = 0;
31
32    // Static energy balance
33    port_b.h_outflow = inStream(port_a.h_outflow) + dh;
34    port_a.h_outflow = Medium.h_default;
35
36    // Static momentum balance
37    m_flow =
38      lambda()*wallFriction_mflow_dp(dp, ...) +
39      (1-lambda())*dp/dp_nominal*
40      wallFriction_mflow_dp(dp_nominal, ...);
41
42    // Heat transfer
43    kc = heatTransfer_kc_mflow(m_flow, ...);
44    effectiveness = 1-exp(-(kc*heatTransferArea/(cp*m_flow)));
45    dh = lambda()*effectiveness*cp*(T_amb - state.T);
46
47    // Auxiliary equations for thermodynamic,
48    // transport properties
49    // ...
50  end Pipe;

```

Listing 6: Pipe model using UTP

Simulating Modelica models with a Stand-Alone Quantized State Systems Solver

Federico Bergero¹ Xenofon Floros² Joaquín Fernández¹ Ernesto Kofman¹ François E. Cellier²

¹CIFASIS-CONICET, Rosario, Argentina

{bergero, fernandez, kofman}@cifasis-conicet.gov.ar

²Department of Computer Science, ETH Zurich, Switzerland

{xenofon.floros, francois.cellier}@inf.ethz.ch

Abstract

This article describes an extension of the OpenModelica Compiler that translates regular Modelica models into a simpler language, called Micro-Modelica (μ -Modelica), that can be understood by the recently developed stand-alone Quantized State Systems (QSS) solvers. These solvers are very efficient when simulating systems with frequent discontinuities. Thus, strongly discontinuous Modelica models can be simulated noticeably faster than with the standard discrete time solvers.

The simulation of two discontinuous models is analyzed in order to demonstrate the correctness of the proposed implementation as well as the advantages of using the QSS stand-alone solvers.

Keywords: *OpenModelica, Quantized State Systems, Micro-Modelica, efficient simulation, discontinuous systems*

1 Introduction

There are numerous reasons to desire efficient simulation of hybrid dynamical systems. Nowadays the attention is focused on various aspects of parallelizing the simulation process, while keeping untouched the heart of any simulation pipeline, namely the numerical solver. Indeed, for most researchers and practitioners, the problem of defining an efficient, general-purpose DAE solver is considered to be solved, with DASSL being the default method for all commercial simulation tools. Besides DASSL, there exists a vast variety of solvers targeting different simulation requirements and families of models.

We argue that the attention should be drawn again

to the "basics" and question the underlying assumption of **time discretization** that traditional solvers use. Already at the end of the nineties, Zeigler introduced a new class of algorithms for numerical integration based on **state quantization** and the Discrete Event Simulation (DEVS) formalism [18]. Improving the original approach of Zeigler, Kofman developed a first-order non-stiff **Quantized State System** (QSS) algorithm in 2001 [16], followed later by second- and third-order accurate non-stiff solvers, called QSS2 [13] and QSS3 [15], respectively. Currently, the family of QSS methods includes also stiff system solvers (LIQSS [17]) as well as solvers for marginally stable systems (CQSS [5]).

There is now plenty of evidence that the QSS solvers offer several advantages over the classical approaches [17, 7, 15, 14]. QSS methods allow for **asynchronous** variable updates, a feature particularly suited to real-world sparse systems where a significant reduction of the computational costs is achieved. Furthermore, QSS algorithms inherently provide **dense output**, i.e., they do not need to iterate to detect the discontinuities. They rather predict them. This feature, besides improving on the overall computational performance of these solvers, enables **real-time simulation**. Finally, QSS solvers come with theoretical global error bounds that other solvers lack [4] and recently parallel version of QSS methods have been developed [3].

Originally, QSS algorithms were implemented under DEVS simulation engines such as PowerDEVS [2]. While these implementations were correct, some features of the DEVS engines introduced a large overhead. Recently, a family of stand-alone QSS solvers were developed in order to overcome this issue [6]. The new solvers achieve a speed-up of one order of

magnitude over DEVS implementations.

The stand-alone QSS solvers simulate models described in a C language interface that contains the ODEs and zero crossing functions as well as additional structural information needed by the QSS algorithms. The C interface can be automatically generated from a simple ODE description by a tool developed for that purpose.

Modelica [10, 11] is a multi-domain, modern language for modeling of complex physical systems. It is an object-oriented language built on acausal modeling with mathematical equations and designed to effectively support modular libraries and a standardized model exchange.

There are various commercial environments, such as Dymola, along with open-source implementations, such as OpenModelica [9], that support the Modelica language specification. All of these tools take as input a Modelica model and perform a series of preprocessing steps (model flattening, index reduction, equation sorting and optimization). An optimized DAE representation of the original system is achieved and efficient C++ code is generated to perform the simulation.

There have been previous attempts to simulate Modelica models with QSS algorithms. In [8, 7] an interface between OpenModelica and PowerDEVS (**OMPD interface**) has been implemented and analyzed taking a first step towards using QSS solvers in the simulation of general Modelica models. The interface allows the automatic transformation of large-scale models to the DEVS formalism in a suitable way, thus enabling simulation in the PowerDEVS environment using QSS methods. However, as this interface uses a DEVS engine it suffers from the previously mentioned overhead issues.

In this work, we extended the OpenModelica Compiler (OMC) in order to automatically translate regular Modelica models into a subset of the Modelica language called μ -Modelica. Then, we developed a tool that automatically generates the C interface structure needed by the stand-alone QSS solver from the μ -Modelica description and simulates it. That way, our work enables Modelica users to exploit the benefits of QSS solvers directly from the OpenModelica environment without any further knowledge, using them just like any other traditional solver.

We also conducted an extensive comparative performance analysis between the QSS solvers and OpenModelica DASSL over two discontinuous models. The results show a noticeable improvement in

terms of simulation time and robustness.

The article is organized as follows: Section 2 provides a brief description of the components needed for the solver. Section 3 uncovers the details behind the implemented stand-alone QSS solver, while in Section 4 specific simulation results of two example models are presented and discussed. Finally Section 5 concludes this study, lists open problems and offers directions for future work.

2 Background

2.1 QSS Simulation

Consider a time invariant ODE system:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t)) \quad (1)$$

where $\mathbf{x}(t) \in \mathbb{R}^n$ is the state vector. The QSS method, [16], approximates the ODE in Eq. 1 as:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{q}(t)) \quad (2)$$

where $\mathbf{q}(t)$ is a vector containing the quantized state variables, which are quantized versions of the state variables $\mathbf{x}(t)$. Each quantized state variable $q_i(t)$ follows a piecewise constant trajectory via the following quantization function with hysteresis:

$$q_i(t) = \begin{cases} x_i(t) & \text{if } |q_i(t^-) - x_i(t)| = \Delta Q_i, \\ q_i(t^-) & \text{otherwise.} \end{cases} \quad (3)$$

where the quantity ΔQ_i is called **quantum**. In other words, the quantized state $q_i(t)$ only changes when it differs from $x_i(t)$ more than ΔQ_i . In QSS, the quantized states $\mathbf{q}(t)$ are following piecewise constant trajectories, and since the time derivatives, $\dot{\mathbf{x}}(t)$, are functions of the quantized states, they are also piecewise constant, and consequently, the states, $\mathbf{x}(t)$, themselves are composed of piecewise linear trajectories.

Unfortunately, QSS is a first-order accurate method only, and therefore, in order to keep the simulation error small, the number of steps performed has to be large.

To circumvent this problem, higher-order methods have been proposed. In QSS2 [13], the quantized state variables evolve in a piecewise linear way with the state variables following piecewise parabolic trajectories. In the third-order accurate extension, QSS3 [15], the quantized states follow piecewise parabolic trajectories, while the states themselves exhibit piecewise cubic trajectories.

QSS methods have Linearly Implicit counterparts (LIQSS1, LIQSS2 and LIQSS3) [17]. The LIQSS methods are explicit (they do not invert matrices or perform iterations) but, under certain conditions, they can efficiently integrate stiff systems.

2.2 Stand-Alone QSS Solvers

The stand-alone QSS solver [6] is a tool that implements the complete QSS family of algorithms without using a DEVS engine.

The tool is composed by two main modules:

1. The simulation engine that integrates the equation $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{q}, t)$ assuming that the quantized state trajectory $\mathbf{q}(t)$ is given.
2. The solvers that given $\mathbf{x}(t)$, effectively calculate $\mathbf{q}(t)$ using the corresponding QSS algorithm.

An important feature of QSS methods is that state variables are updated at different times. Thus, at each simulation step, only some components of $\mathbf{f}(\mathbf{q}, t)$ are evaluated. In consequence, the simulation engine requires the model to be described so that each component of $\mathbf{f}(\mathbf{q}, t)$ can be evaluated separately. Similarly, each zero crossing condition must be given by a separate function together with the corresponding event handler. In addition, structural information describing the dependencies between variables and equations must be provided.

All the simulation framework, including the simulation engine, the solvers and the models are written in plain C.

Since it is very uncomfortable for an end-user to describe a model providing all this information, the QSS solver tool includes a translator that generates the C interface with all the structural information from a regular ODE description.

This ODE description can have the following components:

- ODEs of the form $\dot{x}_j = f_j(\mathbf{x}, \mathbf{a}, \mathbf{d}, t)$ where \mathbf{x} are continuous state, \mathbf{a} are algebraic and \mathbf{d} are discrete state variables
- Algebraic equations of the form $a_j = g_j(\mathbf{x}, \mathbf{a}, \mathbf{d}, t)$ with the restriction that a_j can only depend on a_1, \dots, a_{j-1} .
- Zero crossing functions of the form $z_j = h_j(\mathbf{x}, \mathbf{a}, \mathbf{d}, t)$.

- Associated to each zero crossing function, two handlers (one for positive and the other for negative crossings) where discrete as well as continuous state variables can be updated.

This description is processed by a parser that computes all the structure, including

- the incidence matrices from continuous and discrete state variables to ODE equations,
- the incidence matrices from continuous and discrete state variables to zero crossing functions,
- the incidence matrices from handlers to ODE equations and zero crossing functions.

This information is then used by a code generator that produces the C interface describing the model.

3 Simulation of Modelica Models with Stand-Alone QSS Methods

As we mentioned above, the stand-alone QSS solver has a tool to extract the structural information from a simple ODE description. In order to exploit this feature, we first developed a language called μ -Modelica and then we extended the stand-alone QSS parser so it understands this language and converts it into the ODE description used by the stand-alone QSS solver.

Then, we extended the OMC so that it generates μ -Modelica models from regular Modelica models.

In this way, regular Modelica models can be automatically simulated by the stand-alone QSS solvers.

In Figure 1 we see the complete compilation and simulation process involved.

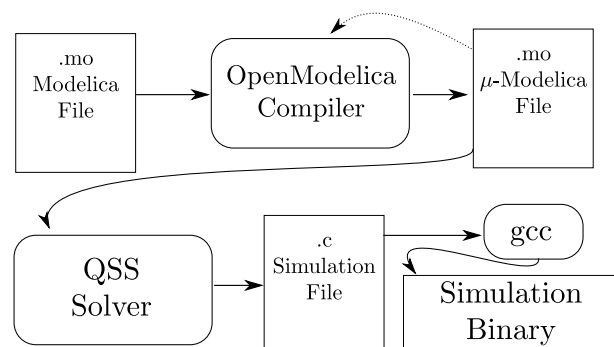


Figure 1: Pipeline of the compilation/simulation process

Below, we first introduce the μ -Modelica language and then we describe the translation process from Modelica to μ -Modelica

3.1 The μ -Modelica subset

The language μ -Modelica was defined to be a subset of Modelica as close as possible to the ODE description accepted by the stand-alone QSS solver. μ -Modelica contains only the necessary Modelica keywords and structures to define an ODE based hybrid model.

The μ -Modelica language has the following restrictions:

- The model is in a flat form, i.e. no classes are allowed.
- All variables are `Real` and there are only three classes of variables: continuous states (`x[]`), discrete states (`d[]`) and algebraics (`a[]`).
- Parameters also belong to class `Real` and they can have arbitrary names.
- Equations are given in explicit ODE form.
- An algebraic variable `a[i]` can only depend on previously defined algebraic variables (`a[1:i-1]`).
- Discontinuities are expressed only by `when` clauses inside the `algorithm` section. Conditions on `when` clauses can only be relations (`<`, `≤`, `>`, `≥`) and, inside the clauses, only assignment of discrete state variables (`d[]`) and `reinit`s are allowed.

This restricted language is not meant to be used by an end user, but only as an intermediate language between OpenModelica and the QSS solver. The end user is supposed to use the complete Modelica language and then use the OMC to get a μ -Modelica file.

3.2 Simulating μ -Modelica models with the stand-alone QSS solver

As we mentioned above, the QSS solver includes a parser that extracts all the structural information from an ODE representation.

This parser was extended in order to understand μ -Modelica language. After this extension, the parser performs the following actions:

- It recognizes Modelica keywords for parameters, and discrete states.

- It takes equations of the form `der(x[i])=expr()`, generating the corresponding ODE and structural information.
- It recognizes clauses of the form `when expr1>expr2 then`, generating a zero crossing function `zc=expr1-expr2` with a handler for the positive crossing containing the expressions that are found inside the clause. If it then finds a clause `elsewhen expr1<expr2 then`, it generates the handler for the negative crossing.
- It also generates the structural information corresponding to the zero-crossing functions and the handlers.

3.3 Converting Modelica models to μ -Modelica

In order to complete the process to simulate regular Modelica models with the stand alone QSS solver, we added a new output target for the OMC to generate μ -Modelica models.

Most of the work is done by what OMC already does without any modification: It first simplifies expressions, sorts the equations and transforms the DAE into an ODE, producing the necessary code for solving the algebraic loops. It also recognizes zero crossing conditions.

Thus, we take the structures generated by OMC and process them as follows:

1. Find the continuous state variables (those where the `der` operator is used), algebraic variables (those solved in the ODE equation that are not states), and discrete state variables (those defined as discrete, including `Integer` and `Boolean` variables.). Boolean variables are replaced by real valued variables where `1.0` is true and `0.0` is false.
2. Parameter names are changed replacing dot(s) for underscore(s). This is done for all identifiers.
3. Continuous state, discrete state and algebraic variables (`Real x[]`, `Real d[]`, `Real a[]`) are defined and code is generated with their initial values.
4. In each equation of the ODE section, each appearance of continuous state, discrete state and algebraic variables is replaced by their corresponding μ -Modelica alias `x[]`, `a[]` or `d[]`.

5. If the equation is part of an algebraic loop, an external solving C function is generated and a call to that function is generated in the μ -Modelica.
6. For each zero crossing function, when and elsewhen clauses are generated. The extra elsewhen is necessary to assign different values to the discrete state variable associated with the crossing function.
7. when clauses are emitted also replacing continuous states, algebraic and discrete state variables in the condition and in the body of the clause.
8. sample operators are expanded using an extra discrete state variable.
9. elsewhen clauses are emitted as regular when in the algorithm section.

For example a model of a bouncing ball in Modelica:

```
model bball1
  Real y(start = 1),v,a;
  Boolean flying(start = true);
  parameter Real m = 1;
  parameter Real g = 9.8;
  parameter Real k = 10000;
  parameter Real b = 10;
equation
  der(y) = v;
  der(v) = a;
  flying = y>0;
  a = if flying then -g else -g -
    - (b * v + k * y)/m;
end bball1;
```

would be translated to μ -Modelica as follows:

```
model bball1
  constant Integer N = 2;
  Real x[N](start=xinit());
  discrete Real d[1](start=dinit());
  Real a[1];
  parameter Real m = 1.0;
  parameter Real g = 9.8;
  parameter Real k = 10000.0;
  parameter Real b = 10.0;
  function xinit
    output Real x[N];
  algorithm
    x[2]:= 1.0 /* y */;
    x[1]:= 0.0 /* v */;
  end xinit;
  function dinit
    output Real d[1];
  algorithm
```

```
    d[1]:= (1.0) /* flying */;
  end dinit;
/* Equations */
equation
  der(x[2]) = x[1];
  a[1] = -d[1] * g + (1.0 - d[1]) *
    (((-b) * x[1] + (-k) * x[2]) / m - g);
  der(x[1]) = a[1];
algorithm
/* Discontinuities */
  when x[2] > 0.0 then
    d[1] := 1.0;
  elsewhen x[2] < 0.0 then
    d[1] := 0.0;
  end when;
end bball1;
```

We see easily that the model has two continuous states, one algebraic and one discrete state variable together with a discontinuity on $x[2]$ that updates the discrete state.

When the original Modelica model contains an algebraic loop, it will be detected by OMC and μ -Modelica will include a piece of code of the form

```
...
function fsolve15
  input Real i0;
  input Real i1;
  output Real o0;
  output Real o1;
  output Real o2;
  external "C" ;
  end fsolve15;
...
equation
...
(a[1],a[2],a[3])=fsolve15(x[2],d[1])
```

together with a C function that solves the loop using GNU Scientific Library (GSL) [12].

This call indicates that variables $a[1:3]$ are computed by a simple C external function, so the QSS parser treats it as a regular function for obtaining the structural information.

In the mentioned external function we improved what was done by OMC taking into account a feature of linear algebraic loops. A linear algebraic equation usually has the form $A \cdot z = b$ (with z being the unknown), where A usually depends on discrete state variables only. Thus, when the change in the continuous state variable only affects the term b , then it is not necessary to invert matrix A in that step.

4 Examples and Simulation Results

In this section we analyze the results obtained using the tools presented in this work.

4.1 Benchmark Framework

As benchmark problems we focused on two systems exhibiting heavily discontinuous behavior, namely a buck converter and a DC-DC buck interleaved circuit. All models were constructed using the Modelica Standard Library 3.1 and can be downloaded from [1].

For each of the examples we used the modified OMC (r11645) to generate the corresponding μ -Modelica model and then the QSS solver to simulate them. In each case, we compare the run-time efficiency and accuracy of the QSS methods against the standard DASSL solver of OpenModelica v1.8.1.

In order to measure the execution time for each simulation algorithm, the reported simulation time from each environment was used. Although OpenModelica provides several ways to measure the CPU time needed for simulation (including a profiler) we observed significant differences in the reported timings. After consulting the OpenModelica developers we finally used `time ./model_executable -lv LOG_STATS` to measure the pure simulation time. We note here that the timing results obtained this way are significantly smaller than the "official" simulation time reported in the OMSHELL or the profiler. Therefore, the speedups we get can be considered to be rather conservative.

Testing has been carried out on a Dell 32bit desktop with a quad core processor @ 2.66 GHz and 4 GB of RAM and in a Intel i7-970 (32 bits) @ 3.20GHz and 2 GB of RAM.

The measured CPU time should not be considered as an absolute ground-truth since it will vary from one computer system to another, but the relative ordering of the algorithms is expected to remain the same.

Calculating the accuracy of the simulations can only be performed approximately, since the state trajectories of the models cannot be computed analytically. To estimate the accuracy of the simulation algorithms for a given setting, reference trajectories ($\mathbf{t}^{\text{ref}}, \mathbf{y}^{\text{ref}}$) have to be obtained. To this end, the LIQSS2 solver was used with a tight tolerance of 10^{-7} .

To calculate the simulation error, each simulated

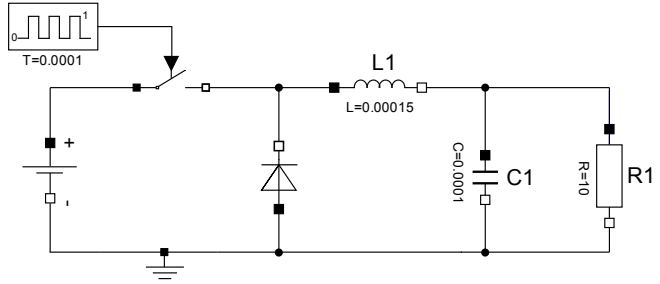


Figure 2: Buck Circuit

trajectory was compared against the reference solution. To achieve this goal, we forced all solvers to output points on the same equidistant grid obtaining simulation trajectories ($\mathbf{t}^{\text{sim}}, \mathbf{y}^{\text{sim}}$) without changing the integration step. Then, the normalized mean absolute error is calculated as:

$$\text{error} = \frac{\text{mean}(|\mathbf{y}^{\text{sim}} - \mathbf{y}^{\text{ref}}|)}{\text{mean}(|\mathbf{y}^{\text{ref}}|)} \quad (4)$$

4.2 Buck circuit

In Figure 2, a DC-DC converter circuit, known as *Buck Circuit*, is sketched. The circuit has two continuous state variables, namely the current through the inductor L1 and the voltage across the capacitor C1. The presence of the switch introduces hybrid behavior to the system. For the simulation error we focus on the C1.V state variable. The model was simulated for 0.01 sec. and the ground-truth trajectory can be seen in Fig 3.

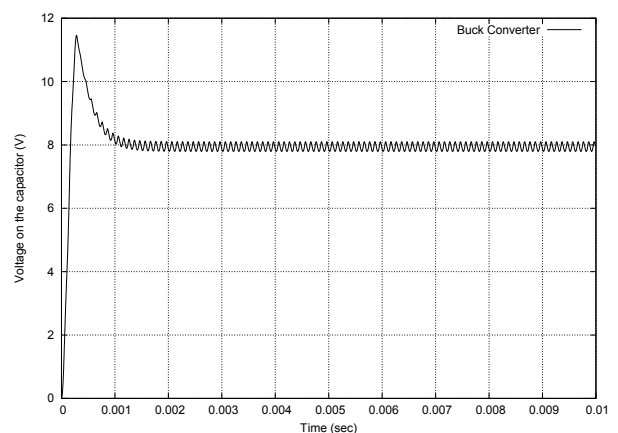


Figure 3: Buck Circuit - Simulation

Initially we simulated the model in OMC using the default number of 500 output points. We observed

that the DASSL solver in OMC fails to detect and handle correctly the events. On the other hand, when we forced OMC to output more points the error decreases because the extra evaluation needed to generate the output forces DASSL to re-evaluate the zero crossing functions, thus detecting the events. This is why we compared OMC's native DASSL solver with different precisions and different number of output points against the QSS solver using the stiff LIQSS2 and LIQSS3 methods. The results are summarized in Table 1.

Indeed we observe that for 500 output points the DASSL solver in OMC doesn't manage to reduce the achieved error when tightening the precision requirements, a clear sign that it fails to simulate correctly the model. When the output points are increased to 10000 the OMC results get closer to the ground-truth trajectory and the error is reduced.

Therefore, it makes sense to compare the runtime efficiencies for the case of 10000 points where we clearly see that QSS methods are more efficient than DASSL in OMC. To perform the simulation for an achieved error of the order of 10^{-5} , LIQSS3 required 12 msec while DASSL needed 74 msec. Therefore, **the use of the LIQSS3 solver instead of the standard DASSL in OpenModelica speeds up the simulation by a factor of 6x.** The achieved reduction in both simulation accuracy and time is depicted graphically in Fig. 4. The results are plotted in a log-log plot where the closer the lines are to the origin the better the corresponding algorithm performs.

Performing an internal comparison between the QSS methods, we see that the third-order LIQSS3 method is slightly more efficient than LIQSS2, especially when the tolerance requirement, thus the achieved error, gets smaller. This is expected, since the LIQSS2 solver needs to take smaller steps compared to LIQSS3 to reach the desired accuracy (e.g. for an error of 10^{-6} LIQSS2 needs 53391 steps while LIQSS3 only used 11314). **Thus, we can conclude that the third-order LIQSS3 algorithm should be preferred for practical applications.** We see also that as QSS algorithms provide dense output, the number of output points does not affect the simulation timings.

Finally, another characteristic of the QSS methods is evident from the obtained results. We verify that in general DASSL performs significantly less steps than any of the QSS methods. However, each one of these steps is much more complicated and time-consuming than the ones performed in a QSS solver, as it in-

volves -in general- estimation of the whole function $\mathbf{f}(\cdot)$. On the other hand, each step in QSS updates one state variable, therefore requiring the evaluation of the corresponding $\mathbf{f}_i(\cdot)$. As the simulated systems get bigger, more complex and sparse, evaluating $\mathbf{f}_i(\cdot)$ is much more efficient than the global $\mathbf{f}(\cdot)$.

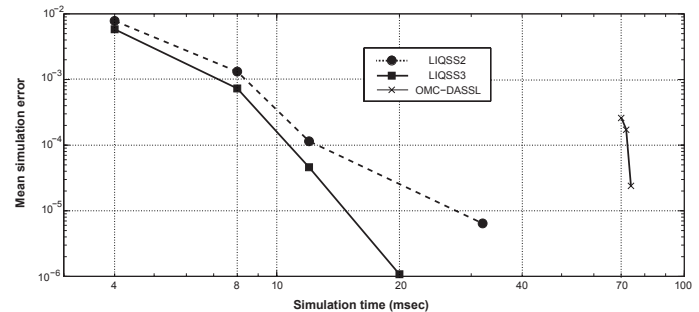


Figure 4: CPU time vs Error for the buck converter model (10000 output points)

4.3 Interleaved DC-DC Circuit

Figure 5 depicts the model of an interleaved buck converter. This circuit is similar to the buck converter analyzed above but it contains several switching sections that are activated at different times in order to reduce the output voltage ripple. In this case, we consider a circuit with four branches.

To build this model, all the components were taken from the MSL 3.1, except for the `booleanDelay` that implements a boolean delay that outputs its received boolean input after a fixed period T . The delay has no memory, i.e. when an input is received, any scheduled output is cancelled and overwritten by the new input.

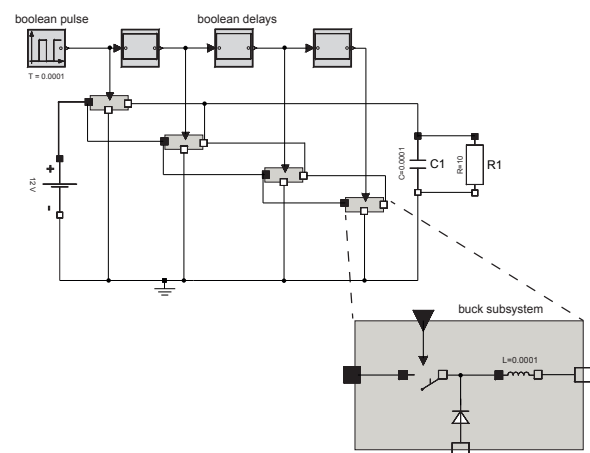


Figure 5: DC-DC interleaved circuit

We have simulated this model for 0.01 sec. again

Table 1: This table depicts the simulation results of various solvers for the buck converter circuit for a requested simulation time of 0.01 sec. The comparison performed includes required CPU time (in msec), number of steps taken, as well as the simulation accuracy relative to the reference trajectory obtained with LIQSS2 and tolerance of 10^{-7} .

			500 output points			10000 output points		
			CPU time (msec)	Steps	Simulation Error	CPU time (msec)	Steps	Simulation Error
QSS	LIQSS3	10^{-2}	4	3351	$5.84E-03$	4	3351	$5.83E-03$
	LIQSS3	10^{-3}	8	4163	$7.31E-04$	8	4163	$7.32E-04$
	LIQSS3	10^{-4}	12	6804	$4.60E-05$	12	6804	$4.61E-05$
	LIQSS3	10^{-5}	20	11314	$1.07E-06$	20	11314	$1.08E-06$
	LIQSS2	10^{-2}	4	3863	$7.83E-03$	4	3863	$7.84E-03$
	LIQSS2	10^{-3}	8	6715	$1.32E-03$	8	6715	$1.32E-03$
	LIQSS2	10^{-4}	12	18519	$1.15E-04$	12	18519	$1.15E-04$
	LIQSS2	10^{-5}	32	53391	$6.42E-06$	32	53391	$6.42E-06$
OpenModelica	DASSL	10^{-3}	22	4273	$3.56E-03$	70	5249	$2.66E-04$
	DASSL	10^{-4}	28	5636	$3.17E-03$	72	5955	$1.75E-04$
	DASSL	10^{-5}	32	7781	$3.28E-03$	74	7623	$2.40E-05$

focusing on the capacitor voltage, getting the simulated trajectory seen in Fig 6. The same experiments as for the buck circuit case were performed and listed in Table 2 where we made the same comparisons as in the previous example (Sec 4.2).

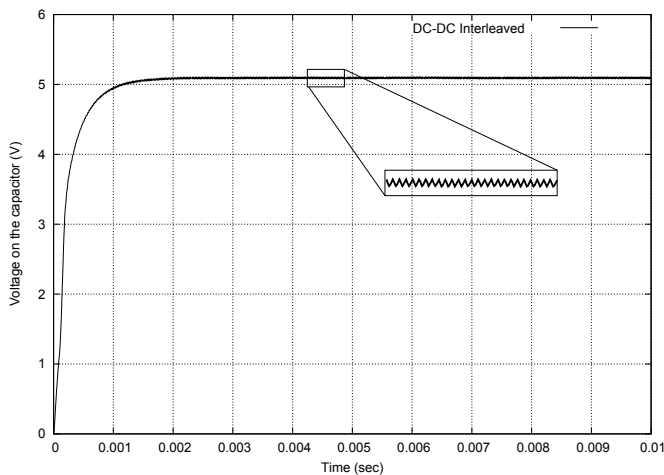


Figure 6: DC-DC Interleaved - Simulation

We see from Fig. 7 that for obtaining a mean error of the order of 10^{-3} OpenModelica's DASSL takes 488 msec while it takes LIQSS2 12 msec and 60 msec for LIQSS3. **This shows 40x and 8x speedups for LIQSS2 and LIQSS3.** The difference in timings between LIQSS2 and LIQSS3 is because the implementation of LIQSS3 is not yet completely optimized and some problems are still present. Also, when asking

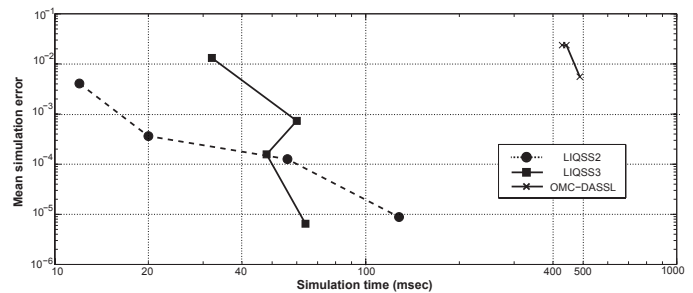


Figure 7: CPU time vs Error for the DC-DC interleaved model (10000 output points)

the QSS solver for 10000 number of output points, neither the error nor the number of steps changes because of the dense output.

In Figure 8 we show the different steady state values obtained with different setups. We see that the discontinuity detection of OMC is heavily influenced by the number of output steps. Here we included Dymola 6.0 result in order to provide a generally-accepted ground-truth solution. We note here that no timing measurements were conducted with Dymola.

5 Conclusion and Future Work

In this article, the integration of the novel stand-alone QSS solvers in the OpenModelica environment is presented and analyzed. The implementation has been tested successfully for both correctness and efficiency in simulating real-world Modelica models.

Table 2: This table depicts the simulation results of various solvers for the DC-DC interleaved circuit for a requested simulation time of 0.01 sec. The comparison performed includes required CPU time (in msec), number of steps taken, as well as the simulation accuracy relative to the reference trajectory obtained with LIQSS2 and tolerance of 10^{-7} .

		500 output points			10000 output points			
		CPU time (msec)	Steps	Simulation Error	CPU time (msec)	Steps	Simulation Error	
QSS	LIQSS3	10^{-2}	32	18396	1.32E-02	32	18396	1.32E-02
	LIQSS3	10^{-3}	60	33426	7.31E-04	60	33426	7.31E-04
	LIQSS3	10^{-4}	48	29408	1.57E-04	48	29408	1.57E-04
	LIQSS3	10^{-5}	64	39951	6.48E-06	64	39951	6.48E-06
	LIQSS2	10^{-2}	12	10715	4.08E-03	12	10715	4.08E-03
	LIQSS2	10^{-3}	20	29082	3.63E-04	20	29082	3.63E-04
	LIQSS2	10^{-4}	56	73218	1.26E-04	56	73218	1.26E-04
	LIQSS2	10^{-5}	128	198001	8.80E-06	128	198001	8.80E-06
OpenModelica	DASSL	10^{-3}	310	14421	4.96E-02	428	17571	2.37E-02
	DASSL	10^{-4}	363	22375	5.03E-02	442	18574	2.37E-02
	DASSL	10^{-5}	496	31387	5.41E-02	488	23625	5.57E-03

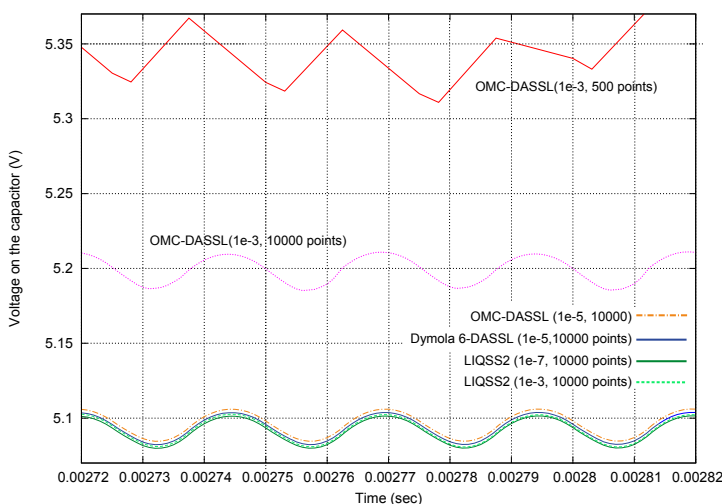


Figure 8: Comparison of the final steady state for different setups

Comparisons on two example models were performed, demonstrating the increased efficiency of the stiff LIQSS solvers over the default DASSL solver of OpenModelica. **Consistent speedups were achieved and the required CPU time was reduced up to 40 times.** Furthermore, for the two systems simulated we observed that the default DASSL solver failed to generate the correct results if we didn't force many output points. Increasing the number of output points, though, means increasing the number of steps taken by the DASSL algorithm, thus the computation time. On the other hand, not only the QSS solvers

simulated correctly the models at all setups but, because of the dense output they inherently generate, the number of steps taken remains constant regardless of how many output points are requested.

However, there still remain open problems to be addressed in the future. First of all, our proposed solution was tested on few examples. A larger set of models has to be simulated and tested for correctness, as well as efficiency, of the implementation. In particular, we should focus on large-scale hybrid models because their dynamics should uncover the power and efficiency of QSS methods. To this end, the μ -Modelica has to be extended to handle more complex systems.

An interesting line of research could be the utilization of the μ -Modelica language as an intermediate language to enable other tools to include Modelica models. Its simplicity makes the burden on the compiler a lot lighter.

The ultimate goal is to integrate the family of QSS solvers (by use of the μ -Modelica translation step) in OpenModelica as native solvers. To achieve this the QSS solver should generate output results in the format expected by the OpenModelica environment. Finally, we need to note that work is also ongoing on improving the QSS solver itself.

6 Acknowledgments

This work was in part funded by CTI grant Nr.12101.1;3 PFES-ES and supported by the OPENPROD-ITEA2 project.

References

- [1] Modelica models for download at. <http://www.fceia.unr.edu.ar/~fbergero/modelica2012>.
- [2] F. Bergero and E. Kofman. Powerdevs: a tool for hybrid system modeling and real-time simulation. *SIMULATION*, 2010.
- [3] F. Bergero, E. Kofman, and C. F. E. A novel parallelization technique for DEVS simulation of continuous and hybrid systems. *Simulation*, 2012. In press.
- [4] F. E. Cellier and E. Kofman. *Continuous System Simulation*. Springer-Verlag, New York, 2006.
- [5] F. E. Cellier, E. Kofman, G. Migoni, and M. Bortolotto. Quantized State System Simulation. In *Proceedings of SummerSim 08 (2008 Summer Simulation Multiconference)*, Edinburgh, Scotland, 2008.
- [6] J. Fernandez and E. Kofman. Implementación autónoma de métodos de integración numérica qss. Technical report, FCEIA - UNR, Rosario, Argentina, 2012.
- [7] X. Floros, F. Bergero, F. E. Cellier, and E. Kofman. Automated Simulation of Modelica Models with QSS Methods : The Discontinuous Case. In *8th International Modelica Conference 2011, Dresden, Germany*, Linköping Electronic Conference Proceedings, pages 657–667. Linköping University Electronic Press, Linköpings universitet, 2011.
- [8] X. Floros, F. E. Cellier, and E. Kofman. Discretizing Time or States? A Comparative Study between DASSL and QSS. In *3rd International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools, EOOLT, Oslo, Norway, October 3, 2010*, pages 107–115, 2010.
- [9] P. Fritzson, P. Aronsson, H. Lundvall, K. Nyström, A. Pop, L. Saldamli, and D. Broman. The OpenModelica Modeling, Simulation, and Development Environment. *Proceedings of the 46th Conference on Simulation and Modeling (SIMS'05)*, pages 83–90, 2005.
- [10] P. Fritzson and P. Bunus. Modelica - A General Object-Oriented Language for Continuous and Discrete-Event System Modeling and Simulation. In *Annual Simulation Symposium*, pages 365–380, 2002.
- [11] P. Fritzson and V. Engelson. Modelica - a unified object-oriented language for system modeling and simulation. In E. Jul, editor, *ECOOP '98 - Object-Oriented Programming*, volume 1445 of *Lecture Notes in Computer Science*, pages 67–90. Springer Berlin / Heidelberg, 1998. 10.1007/BFb0054087.
- [12] M. Galassi. *GNU Scientific Library Reference Manual*, third edition, 2009.
- [13] E. Kofman. A Second-Order Approximation for DEVS Simulation of Continuous Systems. *Simulation*, 78(2):76–89, 2002.
- [14] E. Kofman. Discrete Event Simulation of Hybrid Systems. *SIAM Journal on Scientific Computing*, 25:1771–1797, 2004.
- [15] E. Kofman. A Third Order Discrete Event Simulation Method for Continuous System Simulation. *Latin America Applied Research*, 36(2):101–108, 2006.
- [16] E. Kofman and S. Junco. Quantized-state systems: a DEVS Approach for continuous system simulation. *Trans. Soc. Comput. Simul. Int.*, 18(3):123–132, 2001.
- [17] G. Migoni and E. Kofman. Linearly Implicit Discrete Event Methods for Stiff ODEs. *Latin American Applied Research*, 2009. In press.
- [18] B. P. Zeigler and J. S. Lee. Theory of Quantized Systems: Formal Basis for DEVS/HLA Distributed Simulation Environment. *Enabling Technology for Simulation Science II*, 3369(1):49–58, 1998.

Fast Simulation of Fluid Models with Colored Jacobians

Willi Braun^a Stephanie Gallardo Yances^b Kilian Link^b Bernhard Bachmann^a

^aUniversity of Applied Sciences Bielefeld, Bielefeld, Germany

^bSiemens AG, Energy Sector, Erlangen

Abstract

The industrial usage of the open-source Modelica tool OpenModelica was limited so far for power plant applications, due to the performance of large fluid systems. This paper presents some efforts to improve the simulation time on benchmark fluid models proposed by Siemens Energy. The main aspects presented here to achieve a faster simulation are an efficient evaluation of the jacobian matrix by a coloring technique, that exploits the sparsity pattern of a modelica model. Therefore the techniques are scratched and applied to benchmark models provided by Siemens Energy.

Keywords: *OpenModelica, Fluid Simulation, Benchmark, Simulation, Jacobian, Coloring, Sparsity-Pattern, DASSL*

1 Introduction

In power plant applications, detailed analysis of the dynamic behaviour of heat recovery steam generators result in very large fluid systems.

Modelica is the preferred modeling language for dynamic simulations within Siemens Energy [5] due to its applicability for multi-domain modeling of physical systems, the high degree of maintainability of Modelica models and the possibility of rapid development of new components in Modelica.

The commercial tool Dymola is mainly used for modeling and simulation. The open-source Modelica environment OpenModelica for industrial and academic usage is getting more and more an alternative and has the large benefit that it is freely available. Fluid modeling with Openmodelica was limited by missing implementation of some special features like Modelica.Media. The OpenModelica compiler flattens now the complete Modelica.Media library. Nevertheless the missing functions are still replaced in all benchmark models by external libraries. In order to make OpenModelica an established Modelica tool, the accuracy and performance have to be comparable with

Dymola.

The aim of the current paper is to present the improvement of the simulation time for special benchmark fluid models using an efficient technique to evaluate jacobians. The benchmark fluid models are developed by Siemens AG, Energy Sector, using the commercial Modelica environment Dymola. Siemens Energy has presented fluid models before, which are suitable for the benchmark of the accuracy and the performance of a Modelica Tool. The complexity of these models have been further refined to build up realistic plant models like used in daily business and to reach model sizes which are suitable for performance tests. On the other hand University of Applied Sciences Bielefeld has developed techniques to generate symbolic jacobians in OpenModelica before ([4],[3]). The derivatives are useful for simulating a model as well as for the sensitivity analysis or the optimization of models. Further, jacobians are necessary to support the next FMI¹ version 2.0 [1]. In the work before it was not possible to show improvements for the simulation. This can be explained mainly by the model size we had tested our implementation on, this was caused by the fact that the generation of symbolic jacobians was not applicable to large scale models. This is solved by generating generic partial derivatives and utilise them to compute the full jacobians. Here we catch up and apply the generation of symbolic jacobians on large scale models provided by Siemens Energy [6].

The paper is structured as follows: In section 2 the usage of the jacobian for the simulation purpose is specified. Further, the coloring and the determination of the sparsity pattern are stated and the application of the coloring to the solving process is described. In section 3 there are given some information about the used benchmark fluid models. Whose performance is measured in section 4. Section 5 summarizes the results of this paper and gives proposals for future work.

¹<http://fmi-standard.org/>

2 Jacobian for Simulation

A Modelica model is typically translated to a basic mathematical representation of differential and algebraic equations (DAEs), before being able to simulate the model. Further, these DAEs are transformed to ODEs (ordinary differential equations) with an algebraic part, which is the starting point.

$$\begin{pmatrix} \dot{\underline{x}}(t) \\ \underline{y}(t) \end{pmatrix} = \begin{pmatrix} \underline{h}(\underline{x}(t), \underline{u}(t), \underline{p}, t) \\ \underline{k}(\underline{x}(t), \underline{u}(t), \underline{p}, t) \end{pmatrix} \quad (1)$$

The jacobian of interest for simulation purpose consists of partial derivatives of the ODE-Block h with respect to the states.

$$J_A = \frac{\partial \underline{h}}{\partial \underline{x}} = \begin{pmatrix} \frac{\partial h_1}{\partial x_1} & \cdots & \frac{\partial h_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial h_n}{\partial x_1} & \cdots & \frac{\partial h_n}{\partial x_n} \end{pmatrix} \quad (2)$$

For solving equation 1 with an integration method like DASSL, the derivatives are needed with respect to the states $\underline{x}(t)$ [7]. After all, DASSL uses the iteration matrix

$$M = \frac{\partial \underline{h}}{\partial \underline{x}} + c j * \frac{\partial \underline{h}}{\partial \underline{x}} \quad (3)$$

for solving a nonlinear system in each step by a modified newton method. This matrix M is almost the same as the partial derivatives with respect to the states beside the $c j * \frac{\partial \underline{h}}{\partial \underline{x}}$ part. But that part is the identity matrix multiplied with a scalar value calculated by DASSL. By default DASSL calculates the iteration matrix M by means of numerical finite differentiation. Therefore it is necessary to evaluate the ODE function h $n + 1$ times. However, it is also possible to equip DASSL with an user-specific routine that provides manually calculated iteration matrix M . Considering issues of performance, the calculation of M is the most critical part. In table 1 are summarized the results for one simulation of two different benchmark models (see 3), where are denoted t_s as simulation time, J_{evals} as number of jacobain evaluations and J_{time} as time of evaluation of the jacobian J_{evals} times. One can see that the calculation of the jacobian matrix takes the major time of the simulation time.

N	x	eqns	N	x	eqns
19	231	942	10	140	826
t_s	J_{evals}	J_{time}	t_s	J_{evals}	J_{time}
10.8	111	9.7	2.4	69	1.4

Table 1: Simulation times vs. Jacobian evaluations

So at that point it's possible to reduce the DASSL solving time. It is quite evident that this could be tackled by exploiting the sparse structure of a Modelica Model. One approach which uses the sparsity pattern to reduce the amount of ODE-function calls is the partitioning of columns in colors and calculating them at once [2]. Additionally the matrix M can be determined in a symbolic way and combined with the coloring approach.

Therefore we test 4 different methods to calculate the jacobians:

- finite difference approximations.
- finite difference approximations with coloring.
- symbolical jacobian generated by OpenModelica.
- symbolical jacobian generated by OpenModelica with coloring.

For the numerical approximation of the jacobian the forward finite differentiation is used, where h is determined by DASSL and it depends on x , \dot{y} , current step size.

$$\dot{y} = \frac{f(x+h) - f(x)}{h} \quad (4)$$

The symbolical jacobians are generated within the OpenModelica compiler (for more details see [3],[1]).

2.1 Coloring Jacobians

The coloring of a matrix means first of all to color columns that have no non-zero-elements in the same row. Thus, the starting point for coloring is the sparsity pattern of a matrix. The determination of the sparsity pattern of a Modelica model is described in the next section 2.2.

Assuming the matrix J with it's sparsity pattern is given as:

$$J = \begin{pmatrix} j_{11} & 0 & 0 & 0 & j_{15} \\ 0 & j_{22} & j_{23} & 0 & 0 \\ j_{31} & j_{32} & 0 & 0 & 0 \\ 0 & 0 & j_{43} & 0 & j_{45} \\ 0 & 0 & 0 & j_{54} & j_{55} \end{pmatrix} \quad (5)$$

In this matrix J for example the columns 1 and 3 and also the columns 2 and 4 have no shared non-zero elements in the rows. Thus, this columns could be calculated at once, since they are structural orthogonal. Finding those structural orthogonal rows could be done by re-formulating the problem as graph coloring of a bipartite graph. The bipartite graph $G =$

$((V_1, V_2), E)$ consists of vertexes V_1, V_2 , where V_1 are all rows and V_2 are all columns. And for every non-zero element an edge e_i is defined between the involved row and the corresponding column, vice versa. For the matrix above the corresponding bipartite graph is drawn in figure 1.

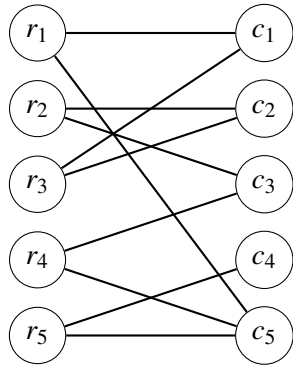


Figure 1: Bipartite graph G

Next step is coloring the column vertexes with the minimum number of colors, so that no row vertex has a connection to columns with the same color. This problem is well-known as NP-hard [2], but for the current purpose it's not very critical to find the optimum, so a fast approximation is well-suited. Therefore a modified partial distance-2 coloring algorithm for bipartite graphs is used as suggested also in [2]. In our tests it reveals a good performance meaning that the solution was really close to the chromatic number $\chi(G, V_2)$, which describes the optimal solution. This observation could be done since there exists a lower bound for $\chi(G, V_2)$. It is also shown in [2] that $\chi(G, V_2) \geq \Delta V_1$ is true. This sounds intuitional for the reason that the minimal partition size depends on the maximum number of non-zero elements in the rows. The time complexity for the algorithm is $O(|E| * \Delta V_1)$, where ΔV_1 is the maximum degree of the vertex $v_i \in V_1$. For example in the jacobian above, it's easy to see that there are several possible solutions as shown in figure 2.

After a coloring C of the columns is found, it's possible to apply it to the calculation of the jacobians. Now all columns with the same color are structural orthogonal and can be calculated at once. Therefore the expected speed up for the calculation is $\text{speedup} = \frac{|V_2|}{C}$.

2.2 Sparsity Pattern

The sparsity pattern for J_A (see equation (2)) of a Modelica Model could also be determined by means of graph theory, because roughly spoken the sparsity

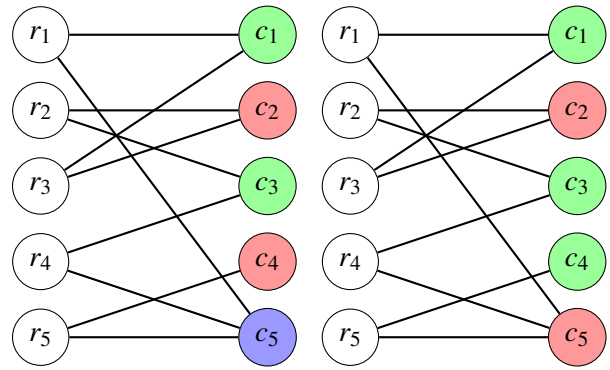


Figure 2: Bipartite graph G

pattern expresses which output variable has a connection to which state. So this could be formulated as a st-connectivity problem in a directed graph. The st-connectivity is a decision problem that asks if the vertex t is reachable from the vertex s . A directed graph is also naturally used in a Modelica tool for the sorting of the equations with the tarjan algorithm. For example if one has a system with 5 equations, and 5 states a directed graph for sorting could look like the one in figure (3).

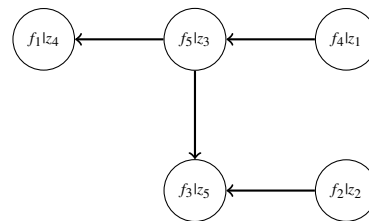


Figure 3: Directed graph for sorting the example system

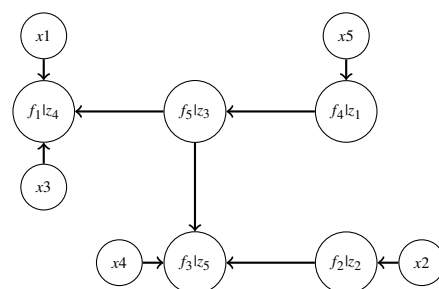


Figure 4: Expanded directed graph for sorting the example system

For the ordinary sorting task by tarjan only the unknowns are considered, since the states are assumed to be known. So for the determination of sparsity pattern one would need to expand the graph by the states. This is done in the way that every equation vertex gets an

additional incoming connection by the states that are present in it. Finally the directed graph could look like the one in figure (4).

The sparsity pattern in equation (6) could than be obtained by finding all reachable vertexes for every state. For every connection that could be found the corresponding element is unequal zero. Finding the reachable vertexes for one state results in one column of the sparsity pattern.

$$J = \begin{pmatrix} * & 0 & * & 0 & * \\ 0 & * & 0 & 0 & 0 \\ 0 & * & 0 & * & * \\ 0 & 0 & 0 & 0 & * \\ 0 & 0 & 0 & 0 & * \end{pmatrix} \quad (6)$$

However, the determination of the sparsity pattern via st-connectivity would require to traverse the whole graph for every state, what is of course not applicable for a large system. Thus one could benefit from the already sorted system and also use additional information from the adjacency matrix. For example consider the following possible sorted adjacency matrix (7) for the system above with the expansion about the states and the equation where they occur.

$$f_i \begin{pmatrix} z_1 & z_3 & z_4 & z_2 & z_5 & x_1 & x_2 & x_3 & x_4 & x_5 \\ \hline f_4 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ f_5 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ f_1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ f_2 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ f_3 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \end{pmatrix} \quad (7)$$

In this BLT-sorted adjacency matrix we consider row after row and propagate the dependent states downwards to every equation. The accumulation of the non-zero elements is arranged in an array of lists for every equation. For the first equation we just add the dependent states x_5 to the corresponding list. For the second equation there are no direct dependencies, but we need to propagate the dependencies for the involved variables. In this case for the variable z_1 which occurs in the first column the lists of f_5 and f_4 are joined. For the next row it is necessary to add the direct dependent variables x_1, x_3 and union them with the indirect dependencies from variable z_3 and so on. This approach results in algorithm with a complexity that depends on the amount of non-zero elements. Our tests indicate even a logarithmic dependence for non-zero elements. Thus the sparsity pattern can be determined efficiently.

3 Benchmark Fluid Models

The first benchmark model (see figure 5) consists of three heated pipes in a row. The first pipe in flow direction is connected to a water source which supplies the liquid flow. The one-dimensional energy, mass and momentum balances are discretized in flow direction. The number of nodes which represent the connection between the discrete elements is N. The heated metal wall of the pipe represents a cylindrical metal wall with L numbers of layers.

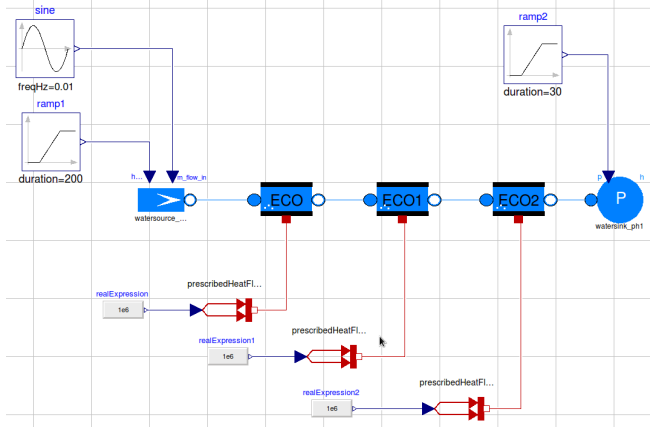


Figure 5: Pipes benchmark model

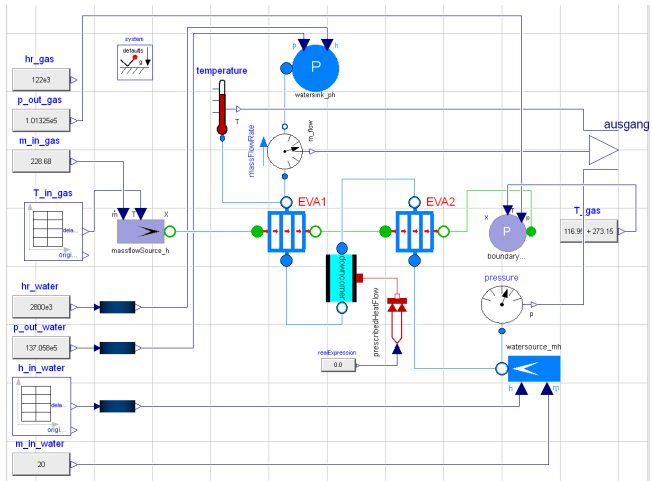


Figure 6: Heat exchanger benchmark model

The central part of our second more complex benchmark is an evaporator model (see figure 6) with parallel tube rows. A parallel flow evaporator consists of several heated tubes connected by an internal splitter at the inlet and an internal mixer at the outlet. For each of the N_l (number of parallel layers) exists a subaggregate which also models the gas-side, using a simple quasi stationary pressure drop. The water and steam flow and the inner heat transfer is modeled using the

	N	x	eqns	colors	N	x	eqns	colors	N	x	eqns	colors
	19	231	942	79	50	603	2430	203	100	1203	4830	403
method	steps	F-Eval	J-Eval	time	steps	F-Eval	J-Eval	time	steps	F-Eval	J-Eval	time
num	922	27184	111	10.8	1014	72854	118	85.3	1058	144874	119	372.3
numC	922	8929	94	4.5	1023	26914	124	38.4	1064	46835	112	144.2
sym	937	1539	103	8.5	976	1643	119	65.2	1052	1732	126	287.2
symC	937	1539	103	4.3	976	1643	119	30.3	1052	1732	126	139.3
Dymola	783	8772	90	1.6	915	23453	106	11.3	1035	43707	103	53.4

Table 2: Simulation time for Tube3Test

	N	x	eqns	colors	N	x	eqns	colors	N	x	eqns	colors
	40	500	2986	95	80	980	5866	175	160	1940	11626	335
method	steps	F-Eval	J-Eval	time	steps	F-Eval	J-Eval	time	steps	F-Eval	J-Eval	time
num	492	38192	75	23.3	537	79131	80	94.7	542	140390	72	436.5
numC	516	10841	106	9.9	505	13810	75	27.3	596	28595	83	152.4
sym	544	774	74	44.9	536	726	77	176.7	556	752	83	792.1
symC	544	774	74	11.9	536	726	77	42.8	556	752	83	206.8
Dymola	359	7306	69	7.36	387	12531	67	22.4	408	23964	69	142

Table 3: Simulation times for HeatExchanger

pipe model. The outer heat transfer is assumed to be constant. This model is suitable for building up huge systems with many states since the number of tube layers of the evaporator can be adapted easily. Compared to a complete and detailed heat recovery steam generator model the model in figure 6 is still small. This justifies the requirement to improve the performance to use in future OpenModelica for power plant simulations.

4 Performance Measurements

The performance measurements are done on a workstation machine (Intel CPU Q9550 @ 2.83GHz). For the time measurements we run the simulation five times take the mean, in addition the initialization process is deducted. Here are depicted the results for the benchmark models 3 with the four modes described above in OpenModelica. Additionally, the results are compared to Dymola. For all simulation was chosen a tolerance of $1e^{-6}$, which is propagate in OpenModelica as absolute and relative tolerance. This may be one reason for the difference in the steps performed by the Integrator.

In the top of the tables 2 and 3 are stated the model details, where the variable N is used for resizing the model resulting in numbers of states, equations for the ODE-function and the colors. The method called

“num” calculates the jacobians numerically and the method “sym” performs it symbolically. The additional “C” marks that the coloring is applied to these methods.

First, it can be stated that the simulation time is effected a lot by the coloring as expected. The factor is a bit lower than expected due to the different number of steps and thus a different number of jacobian evaluation in each simulation. This can be considered as numerical artefacts which are propagated and then induce small differences in the step-size chosen by the integrator. This effect can't be observed for the symbolic solution. Further, it can be stated for the numerical solution the amount of ODE-function evaluation is reduced dramatically and it tends to be close to Dymola. This suggests that Dymola uses a similar techniques.

5 Conclusions

The aim of this paper was to show that one key element for a Modelica Tool to perform a fast simulation is the exploiting of the sparsity pattern for the determination of jacobians. Therefore it is necessary to determine the sparsity pattern and partition the jacobians calculation in order to reduce the evaluation time. This is realized by graph theoretical means in OpenModelica. Further it was shown on the presented benchmark models that

the effect is significant, moreover this feature pushes OpenModelica further to an efficient simulation environment for relevant industrial problems.

Acknowledgments

The German Ministry BMBF has partially funded this work (BMBF Förderkennzeichen: 01IS09029C) within the ITEA2 project OPENPROD (<http://www.openprod.org>).

References

- [1] Åkesson J, Braun W, Lindholm P, Bachmann B. Generation of Sparse Jacobians in the Function Mock-Up Interface 2.0. In: Proceedings of the 9th Modelica Conference, Munich, Germany, Modelica Association, 2012.
- [2] Assefaw H. Gebremedhin, Fredrik Manne, and Alex Pothen. What color is your jacobian? graph coloring for computing derivatives. *SIAM Rev.*, 47(4):629–705, 2005.
- [3] Braun W, Ochel L, Bachmann B. Symbolically Derived Jacobians Using Automatic Differentiation - Enhancement of the OpenModelica Compiler. In: Proceedings of the 8th Modelica Conference, Dresden, Germany, Modelica Association, 2010.
- [4] Fritzson P. et. al.: OpenModelica System Documentation, PELAB, Department of Computer and Information, Linköpings universitet, 2010.
- [5] Siemens Energy: <https://www.energy.siemens.com>
- [6] Link K, Vogel S, Mynttinen I. Fluid Simulation and Optimization using Open Source Tools. In: Proceedings of the 8th Modelica Conference 2010, Dresden, Germany, Modelica Association, 20th to 22nd 2011 2010.
- [7] Petzold L. R.: A Description of DASSL: A Differential/Algebraic System Solver, Sandia National Laboratories Livermore, 1982.

Modelling and Calibration of a Thermal Model for an Automotive Cabin using HumanComfort Library

Stefan Wischhusen
XRG Simulation GmbH
Harburger Schloßstraße 6-12, 21079 Hamburg, Germany
wischhusen@xrg-simulation.de

Abstract

This article aims to describe a modular system level modeling approach for the thermal behavior of an automotive cabin. The model is parameterized with geometric and physical data. At the end a set of 6 parameters is used to calibrate the model with two measurement data sets: one for a passive heat up and active pull down and one for a cold heat up. The procedure can be used as a recipe for developing own models of the same kind which may be used in integrated thermal management studies.

Keywords: automotive cabin; calibration; thermal simulation; air conditioning; integrated thermal management

1 Introduction

The assessment of the thermal behavior of an automotive cabin as a part of the whole vehicle becomes more and more important while the air conditioning system is not just responsible to cool and heat the passenger compartment but also has to condition other sensitive parts. Alternative, energy-saving vehicle concepts require innovative concepts to manage multiple heating and cooling loads. This has to be achieved by thoroughly optimizing many factors: e.g., energy consumption, component utilization as well as life-time reduction and last but not least passenger comfort.

Simulation models are required to allow a system-wide analysis on a conceptual level. The Modelica technology offers a multi-domain platform which allows users to combine different physical systems in order to predict their interaction. Such a configuration is for instance given by a combination of air conditioning cycle, air distribution system and cabin.

Automotive air conditioning cycles were modeled using Modelica since 2000 using different free and commercial libraries (ThermoFlow, ThermoFluid, ACLib [2], AirConditioning [3] and TIL by TLK-Thermo GmbH). The AirConditioning library is used by many European companies since 2004 and has become a standard tool for German automotive companies. In order to model the interaction between the vapor compression cycle and the cabin a modular and flexible model for the cabin was missing, though. Therefore, XRG Simulation decided to develop such an approach in the EuroSysLib-D project [1] which is provided by the resulting HumanComfort library. This model can be directly connected to open Modelica.Fluid air distribution models or to the AirConditioning library (version 1.8 and higher).

Tools for the thermal simulation of automotive cabins are THESEUS-FE [4], EXA PowerFLOW and PowerTHERM, which use CFD approaches for their models. Those models easily count up to some 10.000 nodes in order to capture the complex interior geometry and the required grid size for transient simulations. A coupling of CFD cabin models to air cycle models is possible by using simulator interfaces like TISC by TLK-Thermo GmbH.

Other system level models were developed by: IFT/TLK-Thermo [5], Baumgart et al. [6], Mezhhab [7] and others. The Modelica model of IFT/TLK-Thermo works with a single air volume and multiple walls and windows. Moreover, the cabin model of Baumgart is using multiple volumes and irradiating numbers for its surfaces.

2 Physical Cabin Modeling

The HumanComfort library[1] enables very flexible modular layouts for modeling physical effects. Any

HumanComfort automotive cabin model may integrate the following physical entities in arbitrary numbers:

- Partitions (opaque or transparent) for multi-layer wall setups
- Air volumes
- Air exchange models and/or flow models
- Internal load models (e.g., passengers, waste heat)
- Irradiation balance models
- Internal surfaces
- Thermal comfort models
- External boundary conditions (e.g., climate, air leakage)

The following physical effects are modeled by the component models of the library:

- Heat transfer by convection
- Heat transfer by conduction
- Heat transfer by direct and diffuse irradiation, distinction between short wave and long wave irradiance
- Convective mass transfer
- Condensation of moisture
- Carbon dioxide emission and balance for recirculation air controls

and wall layers depends on the desired resolution for temperature (and other states). Focusing on air temperatures the following layouts are appropriate:

- Single air volume for pure convective driven simulations (e.g. during air conditioning operation)
- 2 air volumes in top/bottom layout if a pre-conditioning of the cabin during which the AC system is switched off has to be simulated

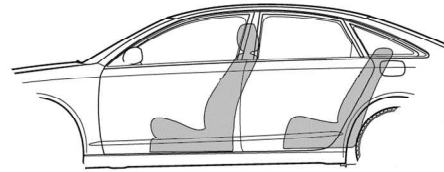


Fig. 2 Automotive cabin layout

A single air volume approach for a very popular middle class sedan car (Fig. 2) is shown in Fig. 1. The model consists of external wall partitions that are exposed to external boundary conditions on the outside for the ceiling, the floor, the left side wall, the right side wall and two smaller parts for the left and right opaque top hull part. Windows are divided into windscreen, two side windows left, two side windows right and rear window. Furthermore, inter-

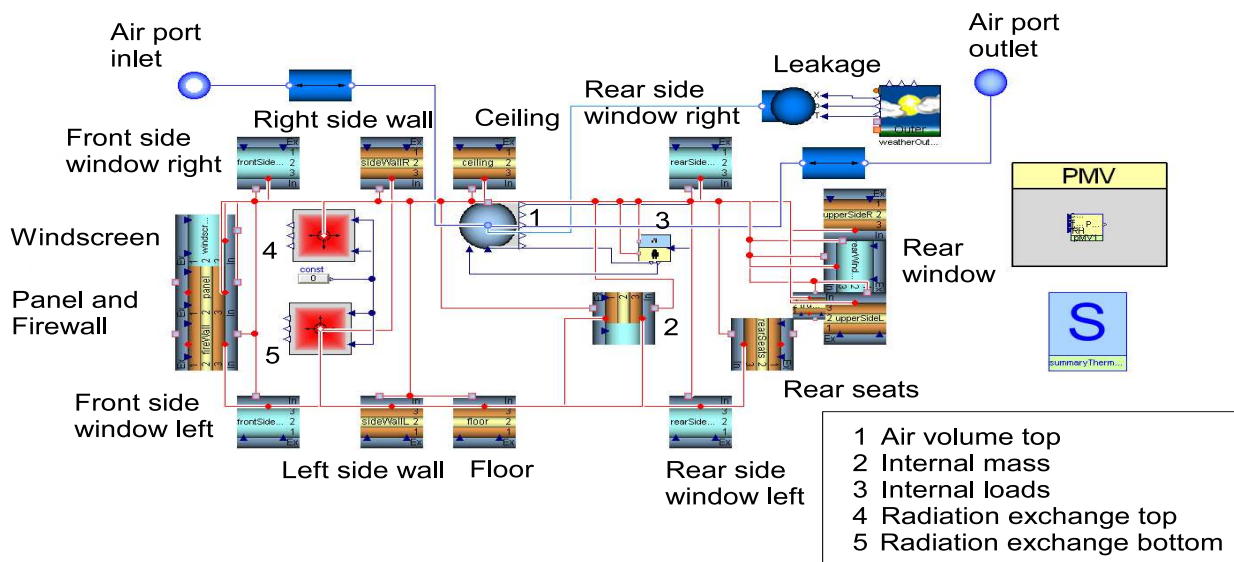


Fig. 1 HumanComfort Modelica cabin model - Single volume approach

The assessment of parameters starts with geometrical parameters. The required discretization of the cabin model with regard to number of air volumes, walls

and wall layers depends on the desired resolution for temperature (and other states). Focusing on air temperatures the following layouts are appropriate:

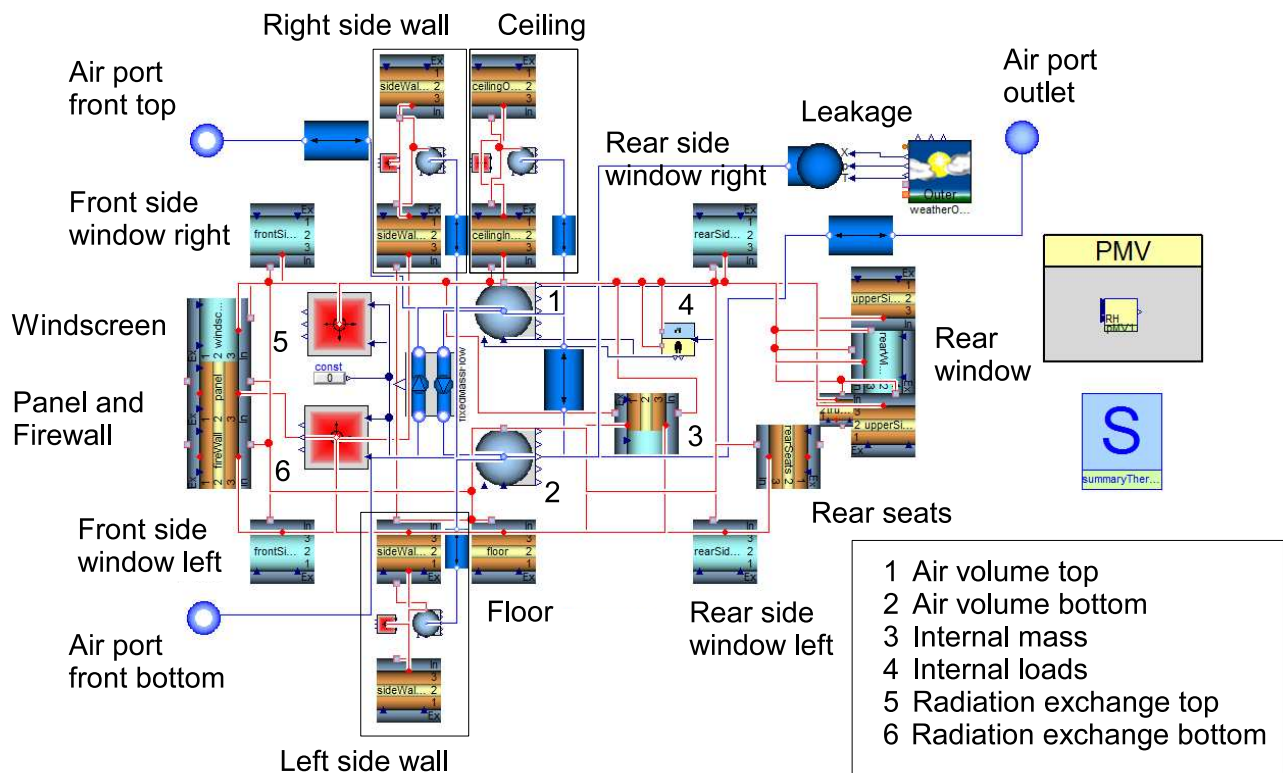


Fig. 3 HumanComfort Modelica cabin model - 2 air volume approach

Since only a single air volume is considered the air distribution modeling is very simple. A single design inlet and a single design outlet were integrated. Please note that a more complex distribution requires additional volumes and flow models that calculate mass flow rates between nodes. Nevertheless, even the single volume approach can be easily extended by more inlets and outlets (see Fig. 3), if required by the measurement setup since the fluid flow connector is according to Modelica.Fluid specification including the stream connector concept.

The 2-volume approach shown in Fig. 3 was created starting from the single volume approach. The upper volume is displaying the air state in the head area of the car. The bottom volume is standing for the average air state in the space below the windowed cabin area. Thus, the convective heat transfer connections of the walls have to consider the location of the partition (top or bottom). The convective heat transfer connectors are represented by the red and gray connectors while the radiation connectors are full red in Fig. 1 and Fig. 3. Additional elements are required for the flow exchange between the top and the bottom volume. For demonstration reasons air spaces have been integrated into the ceiling and side walls in order to simulate air temperatures here as well.

A partition is modeled as a flat but inclined wall with one-dimensional parameters. It may consist of up to 9 layers with independent properties. The outer heat transfer is due to irradiance and convective heat transfer. Optional one can also determine a heated-layer for wall heating (e.g. for seat heating). Fig. 4 explains how geometry parameters are specified. The azimuth angle of a wall describes the horizontal direction of the **outside, ambient** surface normal. A south bound direction is defined to have an angle of 0° . Furthermore, the user has to specify the tilt angle (or zenith angle) between horizontal plain and the walls surfaces. If the zenith angle is 0° or 180° the azimuth angle is meaningless. For surfaces with significant curvature it is straightforward to separate the wall section into parallel partition models.

It is usually not easy to determine properties of the multi-layered cabin walls. Another challenge is defined by the later calibration of the cabin model since the physical parameters of each layer are potentially uncertain. Therefore, it may be a better approach to calculate average properties for a compound of materials and calibrate three property parameters for a wall.

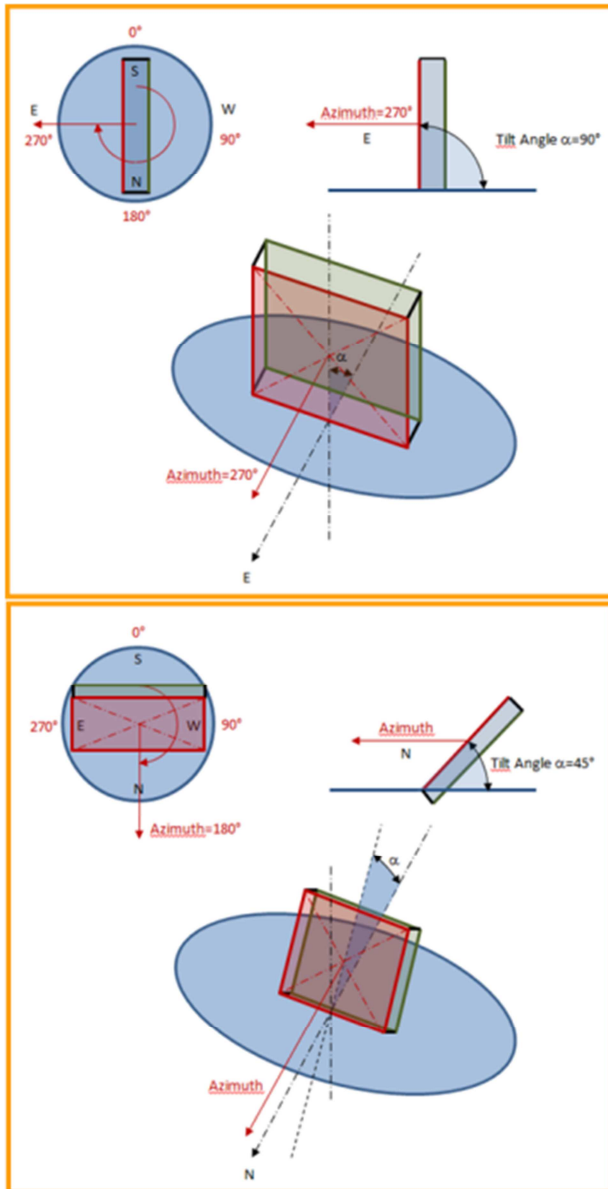


Fig. 4 Partition model orientation parameterization

The average heat capacity can be found by:

$$c_{avg} = \frac{\sum c_i \cdot M_i}{\sum M_i}$$

with

- c_{avg} average heat capacity [J/(kg.K)]
- c_i specific heat capacity for material fraction [J/(kg.K)]
- M_i mass of material fraction [kg]

The average density with respect to thermal behavior follows from:

$$\rho_{avg} = \frac{\sum M_i}{V_{tot}}$$

with

- ρ_{avg} average density [kg/m³]
- V_{tot} total volume of compound [m³]

The average thickness of the compound in the sense of heat conduction is defined by:

$$s_{avg} = \frac{V_{tot}}{A_{HT}}$$

with

- s_{avg} average thickness of compound [m]
- A_{HT} projected heat transferring area [m²]

In order to determine the average heat conductivity of a compound one can choose from two approaches that will be detailed subsequently.

Approach 1 “ideal one-dimensional layers“

$$\lambda_{avg} = \left(\frac{1}{\sum \frac{s_i}{\lambda_i}} \right) \cdot s_{tot}$$

with

- λ_{avg} average heat conductivity [W/(m.K)]
- λ_i heat conductivity of material fraction [W/(m.K)]
- s_i average thickness of one ideal layer [m]
- s_{tot} total thickness of compound [m]

Approach 2 “measurement“

$$\lambda_{avg} = \frac{\dot{Q}_{oi}}{A_{HT} \cdot (T_o - T_i)} \cdot s_{avg}$$

with

- \dot{Q}_{oi} heat flow rate from outside to inside [W]

T_i	Average inside surface temperature [K]
T_o	Average surface temperature on outside [K]

It has to be pointed out that the main contribution to the heat conduction of cabin hulls is defined by insulations and air. A detailed model of air gaps inside doors and ceilings is also possible to create with HumanComfort library (see Fig. 3).

Typical values for materials found in automotive cabins are listed in the Tab. 1 below.

Tab. 1 Material thermal property data

Material	ρ [kg/m ³]	λ [W/(m.K)]	c [J/(kg.K)]
Tin (Steel)	7800	58	480
Insulation	60	0.047	1680
Carpet	750	0.072	1000
Glass	2500	1	800
Plastic	1300	0.21	1470

Window partitions are characterized by further parameters for emissivity and absorption of irradiance. Those parameters are usually well known although there might be also manufacturers who provide the solar heat gain coefficient (SHGC) instead. This factor does not distinguish between the temporary internal and external heat transfer coefficients which is a problem, when boundary conditions change. Thus, the experimental heat transfer coefficients have to be known in addition.

Typical values for short wave transmission factors τ_{sw} , short wave absorption factors α_{sw} and long wave emission factors ϵ_{lw} of different single pane glasses are given in Tab. 2 (refer also to [8]). Those factors have a considerable impact on the heat load of a cabin.

Tab. 2 Window irradiance transmission, absorption and emission data

Glass	τ_{sw} [-]	α_{sw} [-]	ϵ_{lw} [-]
Clear	0.84	0.08	0.91
Green	0.60	0.32	0.80
IR	0.50	0.41	0.80

Such factors are also required for opaque internal and external surfaces as part of the cabin hull (see Tab. 3).

Tab. 3 Hull surface irradiance factors

Hull	α_{sw} [-]	ϵ_{lw} [-]
White	0.30	0.85
Dark blue	0.80	0.90
Black	0.99	0.98
Internal	0.80	0.80

2.1 Boundary Conditions for Simulation

Measurements from an experimental facility were supplied for two experiments at different boundary conditions:

1. Passive heat up and pull down scenario at 45°C ambient temperature and 1000 W/m² vertical, direct irradiation (summer), refer to Fig. 5 to 7
2. Heat up scenario at -20°C ambient temperature (winter), refer to Fig. 8 to 10

The passive heat up and active pull down scenario starts with a passive preconditioning of the cabin. This is achieved by radiant heaters installed above the cabin. After one hour of heating the driver enters the car and starts the engine as well as the AC system. The driving cycle started after the preconditioning consists of three speed intervals: 1. 32 km/h, 2. 0 km/h (idle), 3. 64 km/h. The driver introduces a sensible heat flow rate of at least 80 W as well as a moisture input of 6.5 g/h.

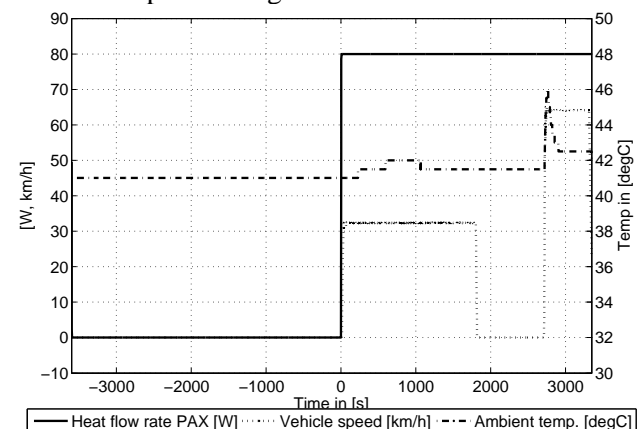


Fig. 5 Boundary conditions for passive heat up and active pull down simulation - Speed, ambient temperature and heat flow rate of passengers

It is important to understand that the car is located in an artificial experimental setup and will not move during all driving cycles. Instead, the air velocity of the surrounding air is changed accordingly. During the passive heat up the vehicle is actually exposed to a small air flux to prevent overheating on some ex-

ternal surfaces. The exact wind speed is unfortunately not known but was assumed to be small. Another large impact on the car's heat balance is imposed by walls of the experimental facility which emit long wave radiation.

The air distribution system of the car is equipped with six measurement sensors for air temperature:

- (Front) Face Center
- (Front) Face Side (Left & Right)
- Face Rear
- Foot Front
- Foot Rear
- Defrost

The mass flow rate of air is recalculated from the fan characteristic assuming a certain fixed distribution between the outlets.

During pull down in summer only the upper outlets are used and foot outlets are closed. The total mass flow rate of air sums up to constant 550 kg/h. Note that the air temperature measured during passive heat up is due to heat dissipation only.

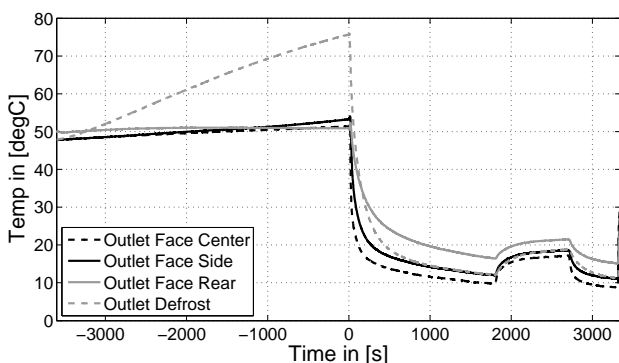


Fig. 6 Outlet air temperatures for passive heat up and active pull down – no outflow for time less than 0 sec.

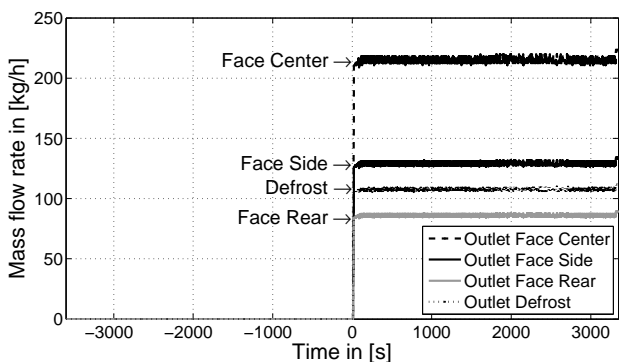


Fig. 7 Outlet mass flow rate of air distribution system for passive heat up and active pull down

In case of the winter scenario a preconditioning of the cabin model is not required, since all partitions are having nearly the same temperature slightly above -20°C . Here, the driving cycle is simpler: 1. 50 km/h, 2. 0 km/h (idle).

Comparing Fig. 9 with Fig. 10 reveals that the air temperature of the rear face outlet is nearly constant until 500 sec although a mass flow rate is shown by the measurement. This deviation from a plausible physical behavior indicates that the Face Rear Outlet is just opened at that time point in order to prevent passenger's exposition to cold draft. It is assumed that the total mass flow rate is correct though. Nevertheless, in order to create correct energy balances it was decided to consider just those outlets which had a temperature larger than -19°C . The total mass flow was evenly distributed across the remaining open outlets.

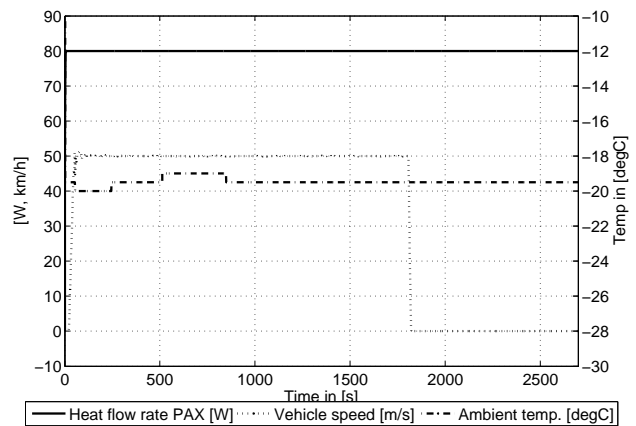


Fig. 8 Boundary conditions for active heat up - Speed, ambient temperature and heat flow rate of passengers

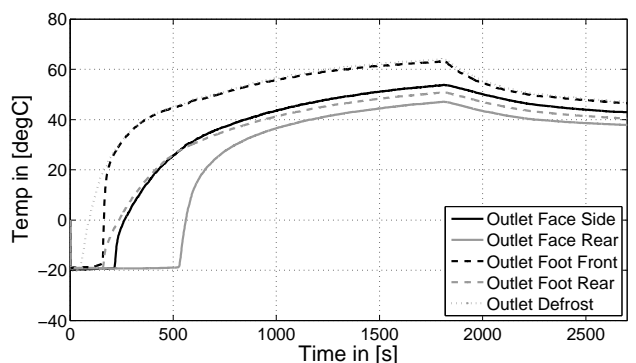


Fig. 9 Outlet air temperatures for active heat up

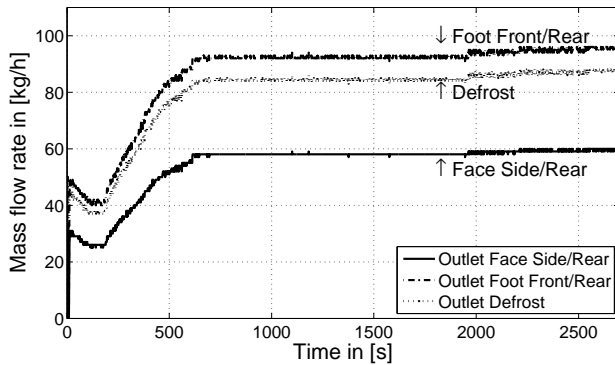


Fig. 10 Outlet mass flow rate of air distribution system for active heat up

3 Calibration Process of the Cabin Model

3.1 Comments on Planning Measurements for Calibrating Cabin Models

There are some pitfalls in using cabin temperature measurements for calibration of cabin models. A general problem is to define correct boundary conditions of the cabin. Especially, the air temperature measurement has to provide at least all temperatures at the virtual outlets of the air distribution system as well as the exhaust/return air inlet since a considerable heat transfer is taking place in the usually not insulated air channels. The effect of temperature gains on the heat load in recirculation mode can be up to 20%.

For multi-volume approaches it is helpful to know approximately the actual mass flow distribution by the air distribution system, since it can become laborious to determine active air outlets at each time point (see section 2.1).

3.2 Calibration Results

The calibration process of the HumanComfort model is required in order to determine important, unknown parameters that have a large impact on the thermal behavior of the cabin. Those are usually:

- Internal and external heat transfer coefficients

- Emission and absorption factors of internal and external surfaces (in this case known)
- The average number of reflections between internal surfaces until the remaining rest of a portion of external short wave irradiance is reflected to ambience (decay of short wave irradiance)

The influence of the cabin hull (ignoring windows) is small on the static heat transfer. Nevertheless the cabin hull walls should not be ignored during transient simulations due to their large heat capacity which causes high heat flow rates to the cabin air in air conditioning or heating mode.

In this study it was possible to calibrate convective heat transfer and heat transfer due to solar irradiation separately since in the winter case simulated on a test facility solar irradiation was not present. Thus, a two step calibration is performed starting with the assessment of the heat transfer coefficients. Afterwards, a calibration of the radiation model's parameters was carried out. In order to simplify the calibration process it was decided to work with average heat transfer parameters. Since there is in all cases a strong variation of air velocity present, a generic heat transfer model in the following form was used:

$$\alpha_{avg\ ext} = \alpha_{const\ ext} + \frac{\alpha_{nom\ ext}}{\gamma_{ext}} \cdot c_{ext}$$

and

$$\alpha_{avg\ int} = \alpha_{const\ int} + \frac{\alpha_{nom\ int}}{\gamma_{int}} \cdot c_{int}$$

The parameters α_{const} and γ were fitted by using the XRG's ModelOptimizer optimization tool to obtain a minimum integral deviation from the average cabin air temperature. ModelOptimizer offers both global and local optimization schemes so that a global optimum can be found.

The external air velocity c_{ext} is equal to the vehicle speed while the internal air velocity c_{int} shows a huge variation across the cabin. In order to simplify the calibration process an average velocity in an arbitrary cross section of the cabin has been chosen:

$$c_{int} = \frac{\dot{m}_{in}}{\rho A_{cross}}$$

The calibration process yielded different values for both measurements due to smaller uncertainties. In the winter case external heat transfer coefficients turned out to be lower than in the passive heat up and pull down case. A very small deviation for both cases was found with the **same** parameters:

- $\alpha_{const} = 7.0$ [W/(m².K)] for internal and external heat transfer,
- $\gamma_{ext} = 0.5$ [J/(m³.K)],
- $\gamma_{int} = 40.0$ [J/(m³.K)],
- Integer number of reflections for short wave irradiance in top node: 1,
- Integer number of reflections in bottom node: 3.

The number of reflections was calibrated by comparison of heat transfer coefficients for different settings in the heat up and pull down case.

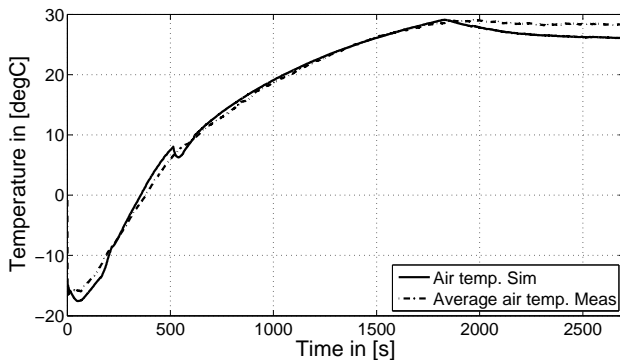


Fig. 11 Calibration result for winter case – comparison of average air temperature

In Fig. 11 and Fig. 12 the result for the average air temperature inside the cabin is shown. The integral deviation of the squared temperature difference is 3790 [K²s] in the heat up case and 6140 [K²s] in the passive heat up and active pull down case, which corresponds to an average deviation of approx. 0.9 to 1.2 K. During heat up the temperature slope is captured in a good way. For both cases there are higher deviations present at the end of each cycle. In the heat up case the temperature deviation starts to increase at 1800 sec when the car speed is decreased to 0 km/h. It was not possible to find heat transfer coefficients that could display the measured behavior though. Thus, this deviation could also be due to wrong assumptions or interpretations of the measurements. In the passive heat up and pull down case a static deviation of approximately 2 K is present in always every speed interval when the AC is on. An exclusive calibration for this case yielded better results with higher heat transfer coefficients. But since both cases required a small deviation average heat transfer coefficients were chosen. Nevertheless, the static temperature deviation is not larger than 1.5 K, again.

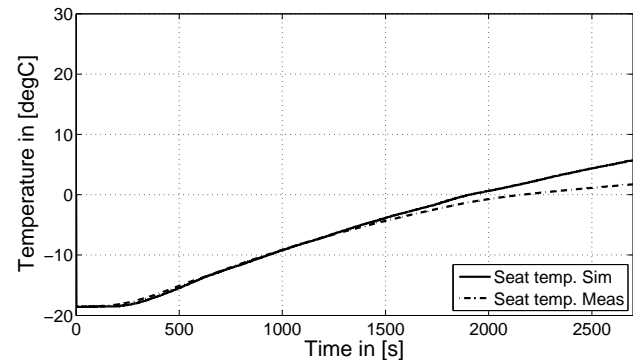


Fig. 13 Seat temperature for the active heat up case

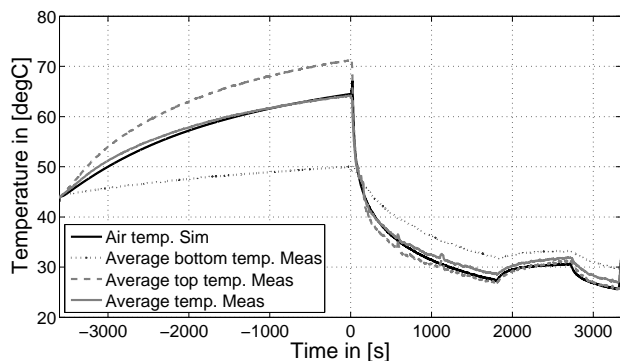


Fig. 12 Calibration result for passive heat up and active pull down – comparison of average air temperature

An interesting auxiliary variable (which was not calibrated) is the seat temperature provided for both measurements (refer to Fig. 13 and 14). The plot reveals that the temperature slope shows in general a comparable plot. It has to be stated that the position of the measurement sensor inside the seat was not known. During the calibration it was found that the internal heat transfer coefficient had a large impact on both plots. The coefficients that were determined at the end of the calibration process yielded a good solution which indicates a successful calibration again.

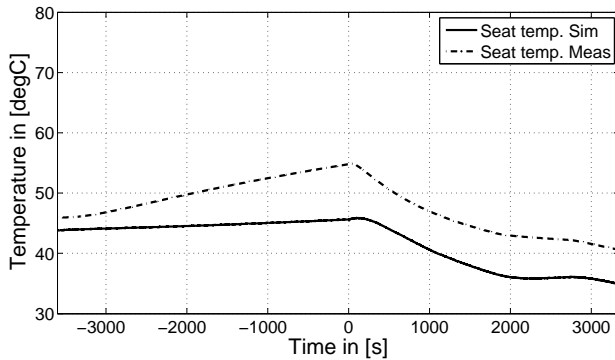


Fig. 14 Seat temperature for the passive heat up and active pull down case

4 Conclusions

The models in XRG's HumanComfort library was successfully used for thermal automotive cabin simulations. The models consider all kinds of thermal heat transfer which is mandatory for using the model in different applications. Due to its modular design the user can easily and quickly exchange components and modify the layout for his needs. Coupling to other Modelica libraries, e.g. for modeling air distribution systems or air conditioning systems is possible since Modelica.Fluid compatible interfaces were used. The modeling process for a single car can be done within one day including parameterization with data provided.

This article was aiming to present a way to calibrate an efficient system level model such that it achieves a comparable accuracy as for more complex approaches (refer to [4]) with much less effort. The time to model the cabin and calibrate it takes approximately two weeks or even less when starting from a template. Furthermore, it was shown that a calibration has to take at least two different cases into account: one case with and one without external short wave irradiance (winter and summer case). The measurements should include a broad range of vehicle speeds and inlet air low rates. With regard to the last point the distribution and amount of air has to be identified as accurate as possible.

5 Acknowledgements

The author would like to thank Denso Automotive Deutschland GmbH for the kind provision of measurement data used in this study.

References

- [1] Michaelsen B., Eiden J.: HumanComfort in Buildings and Mobile Applications: In proceedings of the 7th Modelica Conference, Como, Italy, 2009, pp. 403-412.
- [2] Pfafferott T., Schmitz G.: Modelling and transient simulation of CO₂-refrigeration systems with Modelica, International Journal of Refrigeration, Elsevier, Volume 27, Issue 1, 2004, pp. 42-52.
- [3] Modelon AB, AirConditioning Library version 1.8, Users Guide, Modelon AB, Sweden, Oct. 2010.
- [4] Neacsu C.-A., Ivanescu M., Tabacu I.: The influence of the solar radiation on the interior temperature of the car, <http://www.theseus-fe.com/downloads>, 2009.
- [5] Kaiser C., Försterling S., Tegethoff W., Köhler J.: Untersuchungen von Regelstrategien für die Omnibusklimatisierung mit Hilfe einer Gesamtfahrzeugsimulation, In proceedings of ASIM GI Workshop, Wolfenbüttel (Germany), Feb. 2012.
- [6] Baumgart R., Tenberge P., Urbanek T.: Senkung des Kraftstoffverbrauchs durch Optimierung der Klimaanlage: In proceedings of 14th international congress and exhibition SIMVEC – Numerical Analysis and Simulation in Vehicle Engineering 2008, Baden-Baden (Germany), 2008.
- [7] Mezhhab A., Bouzidi M.: Computation of thermal comfort inside a passenger car compartment, Journal of Applied Thermal Engineering, Elsevier, Volume 26, 2006, pp. 1697–1704.
- [8] Grossmann, H.: PKW-Klimatisierung, Springer Verlag, Berlin, 2010.

Holistic Vehicle Simulation using Modelica – An Application on Thermal Management and Operation Strategy for Electrified Vehicles

Claude Bouvy
Forschungsgesellschaft Kraftfahrwesen mbH
Steinbachstraße 7, 52074 Aachen
bouvy@fka.de

Sidney Baltzer Peter Jeck Jörg Gissing Thomas Lichius Lutz Eckstein
Institut für Kraftfahrzeuge – RWTH Aachen University, Aachen
Steinbachstraße 7, 52074 Aachen
baltzer@ika.rwth-aachen.de

Abstract

The increasing electrification of the drive train in the automotive environment leads to higher requirements for automotive systems and their design. Therefore, a computer based methodology to support the engineer in the design phase of car concepts, components and control algorithms is desirable. All relevant sections of a vehicle development process, e.g. longitudinal and lateral dynamics, thermal management or the power supply should be considered. Due to this necessity a new holistic vehicle library is developed at the Forschungsgesellschaft Kraftfahrwesen mbH Aachen (fka) and Institute of Automotive Engineering (ika) of RWTH Aachen University. The introduced holistic method is applied exemplarily on architecture with the traction battery as thermal storage to determine the potential of such a design on the overall efficiency and to analyse different operational strategies.

Keywords: thermal management; vehicle simulation; traction battery, electric vehicle, range extender, thermal storage, control strategy

1 Introduction

Due to ecologic and economic reasons, the overall efficiency and the emissions, both local and global, of individual mobility have to be improved. An increased electrification of the drive train is currently being considered as a promising approach for reducing both the energy demand and the emissions. However, an increased electrification of the drive

train, i.e. replacing or partly substituting the internal combustion engine, implies the integration of new components as well as a higher number of energy conversion units.

The augmented number of components, as well as their diverging requirements and operating conditions will clearly increase the complexity of electrified car architectures. On the thermal side for example, the integration of temperature sensitive components, e.g. lithium ion batteries, may imply more complex cooling circuit architectures, as the relevant operating temperatures clearly differ to those of an electric machine or an internal combustion engine. On a mechanical level for example, there are several possibilities to couple an internal combustion engine and an electric machine: e.g. parallel and serial hybrids.

Furthermore the increased efficiency of the electric machine compared to the internal combustion engine, will also increase the complexity of both the architecture and the operation strategies. For battery electric vehicles (BEV) for example, the cabin has to be heated by means of electric energy, as in general no or little waste heat is available at a sufficiently high temperature level. Thus, for highly electrified concepts the cabin heating will directly influence the drive train, the power net and the design of the control strategies. To minimise the used electric energy heat pump systems and improved heating control strategies are possible alternatives (cf. e.g. [1]).

The given examples clearly show that a strongly increased complexity has to be expected for the design phase of future cars. Currently an overall design ap-

proach is missing. In general different and mostly incompatible tools are applied for different design tasks and the overall design process is strongly hierarchic. Up to now such a top-down approach was practicable, as the correlation of the energy flows was minor. In general the internal combustion engine, as the core energy conversion unit, implicated the design of most other units, e.g. the cooling circuit.

Furthermore, the different energy forms, chemical, mechanical, electrical and thermal, are increasingly correlated for electrified car concepts. The higher complexity as well as the necessity of a holistic approach requires new tools to support the engineer in the design process.

2 Library description

The holistic model library developed at Forschungsgesellschaft Kraftfahrwesen mbH Aachen (fka) and Institute of Automotive Engineering (ika) of RWTH Aachen University (cf. [2]) takes into consideration all energetic (mechanical, electrical, thermal and chemical) and logical (sensors, actors and control units) flows including dynamic boundary conditions (e.g. drive cycles, ambient conditions) of automotive concerns. It follows a layer based level approach. Basically the modeling library is structured as illustrated in Fig. 1.

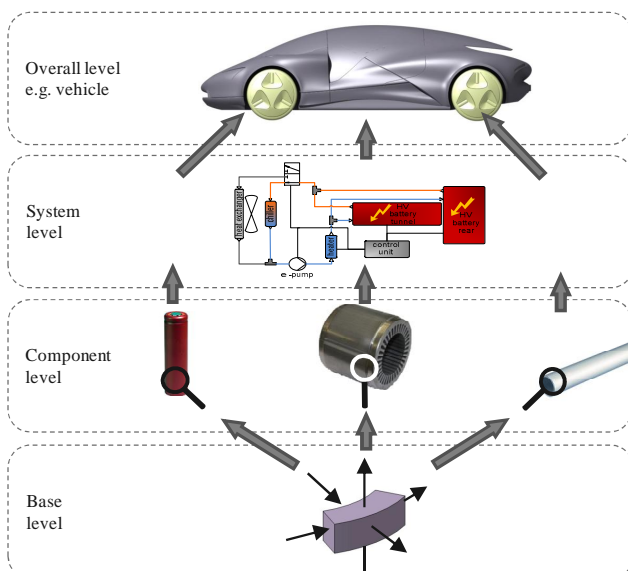


Fig. 1: The four level structure of the holistic tool

2.1 Base Level

At the lowest level generalized elements are implemented which can easily be adapted due to the object oriented modeling property of inheritance or instantiation. On the base level the following packages are implemented. All elements on that level are not computable and are combined later on the components level.

- ThermalLib
- ElectricLib
- MechanicLib
- StateModelLib
- Utilities

The ThermalLib contains all base classes of thermal concern. Based on a general volume element with generalized mass and energy balances and properties, a fluid and a solid element are derived and used for all calculations. Secondly the geometric information of these elements is defined.

For a fluid element the dynamic momentum balance is calculated. A variable modeling depth of pressure drop calculation method may be adapted by choosing a flow model. A heat transfer model calculates the coefficient of heat transfer and provides the necessary interface to e.g. the surrounding ambient. For the solids variable geometries are implemented based on a solidElement, so that new models can easily be generated on the components level. This is illustrated in Fig. 1 where a standardized shell element is used for electric machine housing, the tube of a heat exchanger or a cylindrical battery cell.

The other packages contain e.g. voltage sources (ElectricLib), inertias (MechanicLib) or general mathematical functions (Utilities).

The StateModelLib uses both a model based and a function based approach, wherein data of literature or specific measured fluids can be chosen.

For physical values thermal, fluid, electric and mechanical connectors are defined using the flow and stream properties. For the logical signals expandable connectors are used.

Interfaces are provided, so that the library stays compatible with the Modelica Standard Library connectors (cf. [3], [4]) and the Vehicle Interface Library (cf. [5]).

2.2 Component Level

At the components level a variable number of base elements are combined to generate models to a chosen level of design. At present the components level has the following structure:

- HydraulicComponents
- DriveTrainComponents
- PassengerCabinComponents

E.g. tubes, valves, heat exchangers or pumps are elements of the Hydraulic Components, whereas gears, clutches, electric machines, internal combustion engines or the traction battery are part of the DriveTrainComponents. The different kinds of car body types are integrated e.g. in the Passenger CabinComponents. All the components inherit from the lower base class level as described above.

Fig. 2 demonstrates the approach of the library by the example of the traction battery. It consists of the electrical model, a thermal model and a Battery management system (BMS). All sub models are implemented as replaceable models. Depending on the issue to be investigated the level of detail may be chosen for the single models. However, for the detailed component design, e.g. the exact shape of the cooling duct of a battery pack, a strongly increased level of detail, i.e. a strongly discretised modelling of the coolant flow, is needed, to judge both the heat transfer and the pressure losses as Thermal model (cf. Fig. 2). For the electric model a modeling approach using manufacture data map or a more detailed calculation on the chemical level may be chosen. The BMS may be simulated as a single Read-Only system or more intelligent systems including a control unit may be chosen.

The single models are linked via standardized connectors. For sensor models expandable connectors of the Modelica Standard Library (cf. [3]) are used.

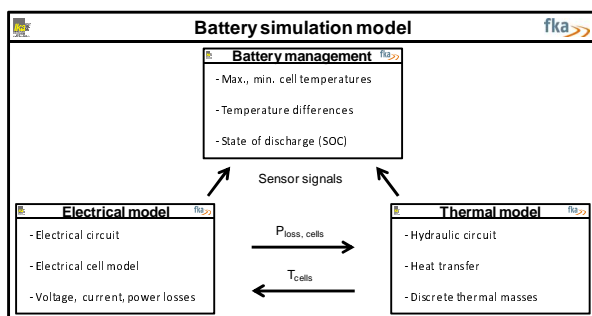


Fig. 2: Modeling approach of the traction battery

2.3 System

At the system level the interactions of energy and signal flow between all components are implemented. The thermal fluid part of the system level is exemplarily shown for the low temperature cooling circuit of a battery electric vehicle in Fig. 3:

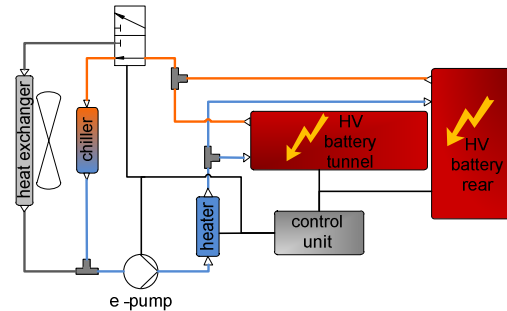


Fig. 3: schematic diagram of a battery cooling circuit

Fig. 4 shows the respective exemplary Modelica model of the configuration, including electrical, thermal and logical signals.

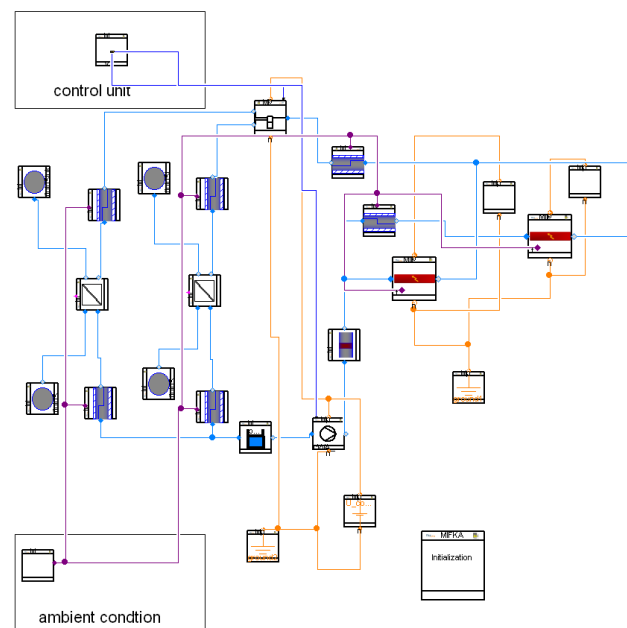


Fig. 4: Exemplary model of the battery circuit

2.4 Overall Level

The vehicle level combines all vehicle sub models such as the power train, the respective cooling circuits, the power supply and the passenger cabin. Beside the global boundary conditions, such as the driving cycle, the route profile, ambient conditions

or initial conditions a control block which consists of the driver and the ECU manages all concerns of control.

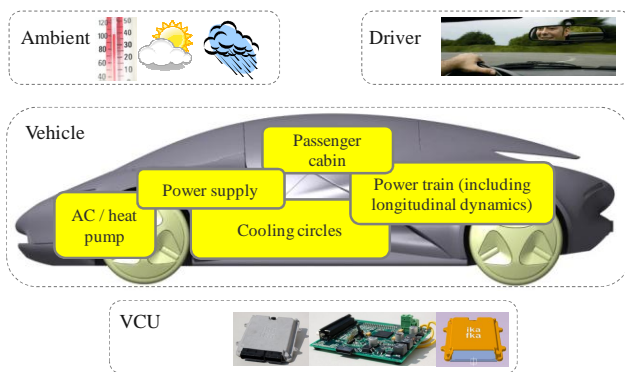


Fig. 5: schematic view of the overall level

3 The traction battery as thermal storage for range extended vehicles

In this chapter an application example is given for the use of the holistic vehicle simulation model approach.

A major challenge for electrified vehicles is to cover the heating demand for the passenger cabin in an efficient way. As stated in Bouvy et al., (cf. [6]) the application of a heat pump system in combination with a preheated traction battery as heat source provides an efficient solution for passenger cabin heating, leading to higher range. In most cases the heat losses of the battery and the thermal capacity are not high enough to cover the heat demand of the passenger cabin so the battery cools down. To avoid an underrun of a critical minimal cell temperature an additional electric heater needs to be switched on so the overall energetic benefit is rather low. Regarding a range extended electric vehicle the waste heat of the internal combustion engine may, besides providing the heat for the passenger cabin, be used to reheat the battery. By this, the overall efficiency of this cogeneration (i.e. producing heat and power) unit may be maximized. Bouvy et al. (cf. [7]) have shown the important benefit of a cogeneration unit on the efficiency of passenger cars.

3.1 System architectures

For this paper two system architectures are discussed for a BEV with a range extender unit.

The first one represents a state of the art range extender design. The cooling circuit of the range extender is connected to the heating and ventilation and air conditioning unit (HVAC) so its waste heat may be used for cabin heating. An additional heat pump system is not considered in this scenario and thermal peak loads are covered by a high voltage electric heater. The operation strategy of the range extender is SOC controlled: it starts when the SOC reaches 20 % and is turned off at a value of 30 % (Charge Sustaining – Mode). The schematic vehicle architecture is illustrated in Fig. 6.

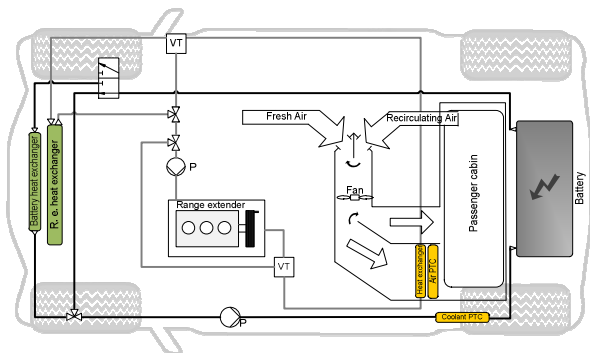


Fig. 6: Vehicle architecture 1

For the second architecture the internal combustion engine cooling circuit is connected to the battery cooling circuit by a fluid/fluid heat exchanger (cf. Fig. 7). Due to this design the battery can be thermally charged. A heat pump system is considered using the battery as heat source to provide an energy efficient heating of the passenger cabin when the internal combustion engine is turned off. The operation strategy of the range extender is thermally controlled by a two level controller. To keep the cell temperatures of the traction battery within an optimal range, the two temperature margins are set to 20°C and 30°C. This operation strategy guarantees high coefficients of performance (COPs) of the heat pump system and an excessive cool down of the battery is avoided.

For both architectures the thermal peak loads of the passenger cabin heating demand are covered by an electric high voltage heater (5 kW).

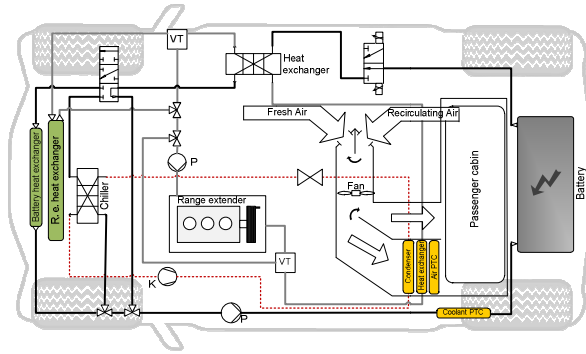


Fig. 7: Vehicle architecture 2

3.2 Simulation setup

In this analysis both layouts have a 44 kW range extender unit, an 80 kW ASM electric engine and a lithium-ion-battery with a nominal capacity of 8.6 kWh (about 40 km BEV range determined on the basis of the NEDC). The data are chosen according to Hartmann and Renner (cf. [8]).

The defined vehicle has a cabin volume of 3 m³ and a surrounding window surface of about 2 m². For the determination of the cabin heat demand a single passenger is assumed and the HVAC is controlled by the passenger cabin air temperature according to Strupp and Lemke (cf. [10]).

All simulations are performed for a Central European winter scenario with an ambient temperature of 0 °C and solar radiation values according to Strupp and Lemke (cf. [10]). At simulation start all thermal masses are in equilibrium at ambient condition. The battery is conditioned to allow regenerative braking immediately at the beginning of the simulation ride (5 consecutive NEDCs). The preconditioning is performed by a 5 kW externally supplied electric heater.

More detailed information concerning model depth and simulation setup can be found in Bouvy et al. (cf. [9]).

3.3 Operational strategies

For the first architecture the battery is electrically charged (SOC= 90%) and thermally preconditioned, so that a min. cell temperature of 5 °C is reached. The internal combustion engine only operates in the “Charge sustaining modus”. The internal combustion engine is operated with the power corresponding to the lowest specific fuel consumption to charge the battery. If an SOC of 30% is reached, the range extender is deactivated (state of the art operation of a range extender).

For the second architecture a thermal operation strategy is applied. The battery is thermally condi-

tioned similar to variant 1 but a lower SOC is chosen to enable electric and thermal charging from the beginning on. A reduced operating power of the range extender is chosen, in order to better fit the power to heat ratio to demand (cf. [7], [11]).

Variant	Operational Strategy	Range Extender Control Strategy
1 st	Without using battery as thermal storage	SOC controlled $P_{\text{mech}} = 19000$ W
2 nd	Using battery as thermal storage Without battery preheating	Thermally controlled $P_{\text{mech}} = 10000$ W

Tab. 1: Investigated variants

In Fig. 8 the Dymola model of the overall system level is shown for the analysed szenarios.

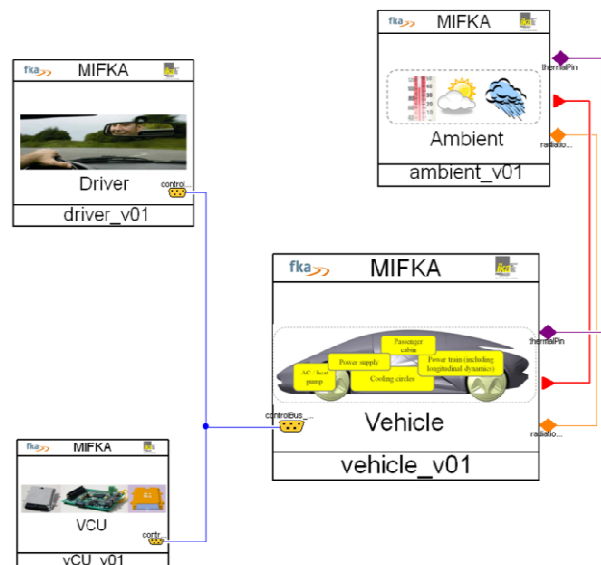


Fig. 8: overall system level in Dymola

4 Simulation Results

Fig. 9 shows the dynamic profile of the average cell temperatures. After the preconditioning phase the thermally operated range extender of variant 2 is turned on. At beginning the thermostatic valve of the internal combustion engine is closed until the ther-

mal masses are heated up. Afterwards the waste heat is used both for cabin heating and to thermally charge the traction battery to a temperature of 30°C. When reaching the threshold the engine is turned off and the heat pump system cools down the traction battery by providing the heating demand for the passenger cabin.

For variant 1 the battery slowly heats up due to charge/discharge losses.

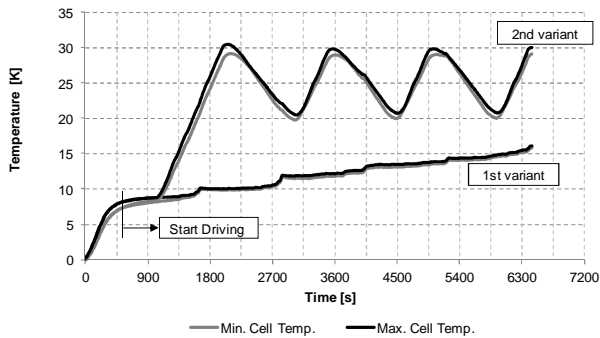


Fig. 9: average cell temperature for the simulated variants

Fig. 10 visualizes the time dependent state of charge curve. Variant 1 is operated purely electrically in the charge depleting mode until the defined SOC of 20 % is reached. Subsequently the range extender is turned on after and the battery is charged again to a SOC of 30 % (charge-sustaining).

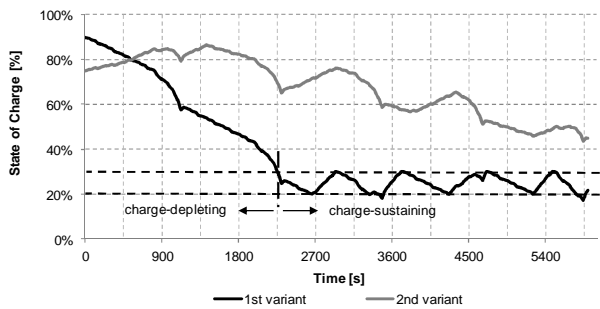


Fig. 10: State of charge of each variant

The operating times vary due to the power requirement for the drive cycle as seen in the velocity profile in Fig. 11.

In the second variant the engine is turned off at about 23 minutes. Over the whole ride the SOC is depleting because of the thermally controlled operation strategy. At the end of the ride the electric charge of the battery remains at about 50 %. Using this strategy the operation intervals of the range extender are

nearly constant except for the first operation interval. Here the battery heating starts from the thermal pre-conditioning level (5°C) and must consequently be operated for a longer time. Afterwards the varying heat transfer due to the vehicle velocity is rather low so a thermally stationary state is reached.

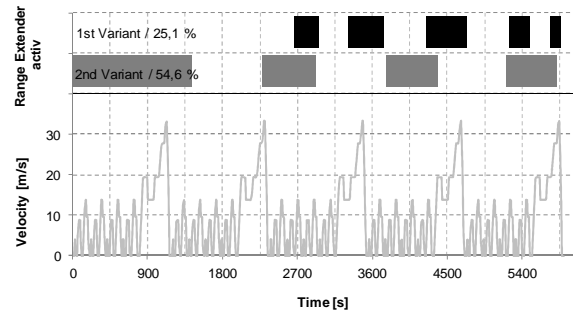


Fig. 11: Range extender operation

Next, the time dependent heat flow rate distributions of the different heating components are discussed. In the analysis it is assumed, that the electrical PTC heater has an efficiency of 100 %, so the electric demand and the heat flow rate are the same. When the internal combustion engine is not operated, its cooling circuit pump is switched off and the remaining heat is not used. This is illustrated in Fig. 12.

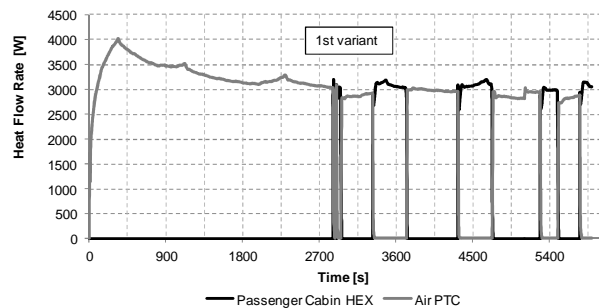


Fig. 12: Passenger cabin heating 1st variant

Regarding variant 2, Fig. 13 shows that due to the high temperature level of the battery’s coolant circuit high COP-values are reached by the heat pump system so an efficient cover of the passenger cabin is achieved while operating purely electrically.

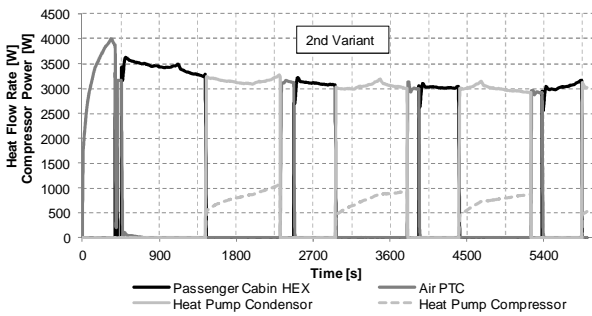


Fig. 13: Passenger cabin heating 2nd variant

Furthermore, an energetic evaluation of both systems is performed. For the sake of comparability, the amount of used primary energy is evaluated for the two variants. In order to evaluate the overall efficiency the overall energy input has to be accumulated. For the discussed variants two different kind of energy forms are used, fuel and electric energy from the grid. For this analysis a primary energy factor of 1.26 is chosen for the fuel (cf. [12]) and 2.6 for electric energy (cf. [13]). This approximately corresponds to the energetic supply situation in Germany.

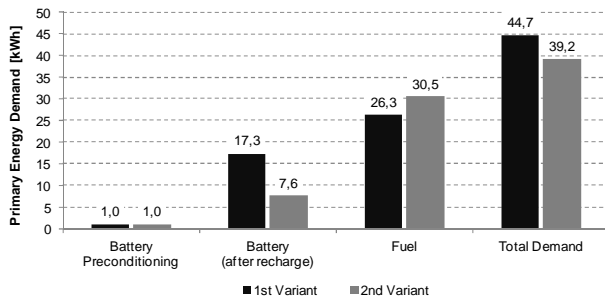


Fig. 14: Primary energy demand for both variants

The results show primary energy saving up to 12 % for architecture 2 in combination with a thermal operational strategy. Thus, for the considered winter scenario the benefit of a cogeneration approach in combination with a heat pump and a thermal storage is clearly stated out.

5 Conclusion

The increasing complexity of actual and future vehicle leads to the need of a holistic modeling development tool taking into account all the classical automotive disciplines such as longitudinal dynamics, electric system or thermal management but also their connection vis-à-vis. Such a holistic library is currently being developed at Forschungsgesellschaft Kraftfahrwesen mbH Aachen (fka) and Institute of

Automotive Engineering (ika) of RWTH Aachen University and was presented in the paper.

An application example was given of the traction battery as a thermal storage of range extended electric vehicles. In the example the benefit of an enhanced cogeneration is shown. A further advantage of such an approach is that the traction battery will mostly be operated in an optimal temperature range and thus, best charge/discharge efficiencies and lifetimes are reached if the range is wisely chosen. However, the influence of this control strategy on the battery's lifetime has to be investigated further on. Due to the scalability of the model library a highly detailed model to determine lifetime strategies of the battery could be chosen for that or/and experimental could be carried out.

References

- [1] M. Jung, A. Kemle, T. Strauss, und M. Wawzyniak, „Innenraumheizung von Hybrid- und Elektrofahrzeugen“, *ATZ - Automobiltechnische Zeitschrift*, Nr. 05/2011, 2011.
- [2] P. Jeck, C. Bouvy, T. Lichius, und L. Eckstein, „Holistic method of thermal management development illustrated by the example of the traction battery for an electric vehicle“, presented at the 20th Aachen Colloquium „Automobile and Engine Technology“, Aachen, 2011.
- [3] Modelica Association, „Modelica® - Release Notes of the Modelica Standard Library Version 3.2“, 2010.
- [4] R. Franke, F. Casella, M. Sielemann, K. Proelss, M. Otter, und M. Wetter, „Standardization of Thermo-Fluid Modeling in Modelica.Fluid“, in *Proceedings 7th Modelica Conference*, Como, 2009.
- [5] M. Dempsey, M. Gäfvert, P. Harman, C. Kral, M. Otter, und P. Treffinger, „Coordinated automotive libraries for vehicle system modeling“, in *Proceedings 5th Modelica Conference*, Vienna, 2006.
- [6] C. Bouvy, P. Jeck, J. Gissing, T. Lichius, S. Baltzer, und L. Eckstein, „Die Batterie als thermischer Speicher: Auswirkung auf die Innenraumklimatisierung, die thermische Architektur und die Betriebsstrategie von Elektrofahrzeugen“, *Wärmemanagement des Kraftfahrzeugs*, Bd. VIII, Essen 2012.
- [7] C. Bouvy, T. Lichius, und P. Jeck, „On the influence of the thermal demand on the overall

- efficiency of future drive train architectures for passenger cars“, *Int. J. Electric and Hybrid Vehicles*, Bd. Vol. 3, Nr. No. 3, 2011.
- [8] B. Hartmann und C. Renner, „Conventional HEV, Plug-In or Range Extender? A conceptual comparison of modern HEVs based on simulations“, presented at the 18th Aachen Colloquium „Automobile and Engine Technology“, Aachen, 2009.
- [9] C. Bouvy, P. Jeck, S. Baltzer, J. Gissing, T. Lichius, und L. Eckstein, „The battery as thermal storage in range extender vehicles: Influence on the architecture and the operating strategy“, in *Hybrid and Electric Drivetrains 2012*, Aachen.
- [10] N. C. Strupp und N. Lemke, „Klimatische Daten und Pkw-Nutzung: Klimadaten und Nutzungsverhalten zu Auslegung, Versuch und Simulation an Kraftfahrzeug-Kälte-/Heizanlagen in Europa, USA, China und Indien“, Frankfurt a. Main, 2009.
- [11] C. Bouvy, T. Lichius, und P. Jeck, „On the influence of cabin heating on the overall efficiency of car concepts“, presented at the 20th Aachen Colloquium „Automobile and Engine Technology“, Aachen, 2011.
- [12] R. Frischknecht und M. Tuchschnid, „Primärenergiefaktoren von Energiesystemen“, Aachen, 2009.
- [13] N.N., *Verordnung über energiesparenden Wärmeschutz und energiesparende Anlagentechnik bei Gebäuden (Energieeinsparverordnung - EnEV)*. 2009.

Modelling of Radiative Heat Transfer in Modelica with a Mobile Solar Radiation Model and a View Factor Model

Arnav Pathak, Victor Norrefeldt, Gunnar Grün
Fraunhofer Institute for Building Physics, Dept. Indoor Climate, 83626 Valley, Germany
arnav.pathak@ibp.fraunhofer.de, victor.norrefeldt@ibp.fraunhofer.de,
gunnar.gruen@ibp.fraunhofer.de

Abstract

This paper presents a model to estimate the solar radiation under clear sky conditions over stationary, moving as well as flying objects. For the latter it is important to predict the peak solar irradiance under clear sky condition to calculate maximum possible solar thermal loading. In this paper results of irradiation over surface on ground and over aircraft windows and windshields at cruise altitude are presented. Another model implemented, calculates the view factor between two or more surfaces. Determination of the long-wave radiant heat exchange between two or more surfaces or heat exchange with a surface itself requires a view factor matrix. There are several analytical solutions available to calculate view factors for simple and known configurations. Many building simulation programs estimate the view factors in a simplified way, especially when complex geometries are involved. The simplified approach may result in high errors of surface temperatures, which can further cause error in energy balance and estimation of comfort level. The purpose of creating this model is to calculate view factors between complex geometries. The view factor matrices of an enclosed space and of a geometry with openings on its surfaces are presented in this paper. A sensitivity analysis of a view factor matrix is also presented.

Keywords: Solar radiation modeling, View factor calculation, Modelica models, Long-wave radiant heat exchange

1 Introduction

Methods to predict solar radiation have a wide range of applications such as:

- Calculation of cooling loads for air conditioners
- Solar heat load on buildings, automobile, aircraft
- Material deterioration under sunlight
- Solar thermal power generation

Absorption and scattering of a solar beam in the atmosphere lead to attenuation of solar radiation. The outer space provides almost complete vacuum due to which there is no attenuation of solar radiation in the outer space. The main sources of absorption and scattering are atmospheric gases and aerosol in the atmosphere respectively. The longer the path travelled by a solar beam through the atmosphere before reaching the surface, the greater is the likelihood that more of it will be scattered or absorbed [1]. Especially for aircrafts the impact of solar radiation at cruise altitude can be high as radiation at cruise altitude can go up to 1200 W/m².

A further objective of this research work is to simulate long-wave radiant heat exchange between complex geometries. Thus the view factors between these geometries have to be determined. Calculation of view factors is a quite complex process, as it requires solving a double area integral. There are several analytical solutions available to calculate view factors for simple and known configurations [2]. In the procedure presented here a pre-processor does the triangular surface meshing and creates a file in stl-format which serves as input for the Modelica model. The results of different geometries are presented in this paper

2 Solar Radiation Model

The solar radiation model can be used to predict clear sky solar radiation over stationary surfaces like building façades or parked automobiles, moving surfaces like vehicles following a predefined path, as well as flying surfaces like aircraft during climb, cruise and descent.

There are two basic models:

- Sun position model
- Surface radiation model

2.1 Sun position model

The sun position model is the global solar model which calculates the position of the sun in the sky at a particular time and at a particular location on the earth [6][7]. This information and the surface orientation are inputs to the surface irradiation model. Both models are set as an inner outer system, so there is no need of physically connecting them.

The input parameters for the global solar model depend on the type of application. If it is a stationary model, then the input parameters are longitude, latitude, altitude, standard time longitude, ground albedo, single scattering albedo, thickness of precipitable water [cm], ozone content of the atmosphere [cm NTP] and forward scatterance. Meaningful default values are implemented to allow simulations even if the user lacks some of this information (see Figure 1). Apart from above mentioned parameters the modeller must select the modelling approach from a drop down menu. There are three modelling approaches implemented to estimate radiation. If visibility data is not available the modeller can select between a modified Pinazo model [3][4] and a hybrid model for estimating global solar radiation [5]. If visibility data is available, the model will calculate angstrom's turbidity from this visibility data. Figure 1 shows the parameter window of the global sun position model for stationary surfaces.

Figure 1 : Parameterization of stationary global sun position model

In case of moving or flying objects, the modeller has to use the mobile global sun position model. As location of a vehicle and orientation of surfaces are constantly changing, this information is set as input. Figure 2 shows the model block of the mobile global sun position model. All the variables changing with time are in the transition profile which is connected to the global model.

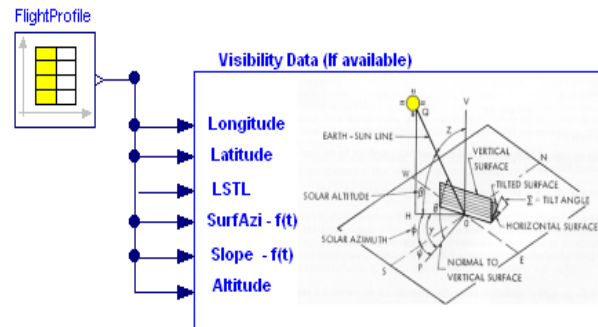


Figure 2 : Mobile sun position model connected with a flight profile

From the location, the day of the year and time of the day, the sun position model calculates the position of the sun in the sky and extraterrestrial radiation. From the altitude and the sun position in the sky, the model calculates the air mass. To determine the solar beam attenuation and irradiation on a horizontal surface, the model calculates absorption and scattering of a solar beam. The direct normal irradiance for a clear sky [3][4] is expressed as

$$I_n = 0.9751 E_o I_{sc} t_r t_a t_w t_o t_g \quad (1)$$

where E_o is the earth's orbit eccentricity correction factor; I_{sc} is the solar constant (1367 W/m^2); t_r , t_a , t_w , t_o and t_g are the transmittance due to Rayleigh scattering, aerosol absorption and scattering, water vapour, ozone and other gases absorptions respectively.

To determine the direct solar radiation on horizontal surfaces using equation (1) it is necessary to know the value of Angstrom's turbidity coefficient. The model implemented will calculate angstrom's turbidity by three different methods. The modeller can select the method from a parameter window. If the horizontal visibility is known, the model computes the value of β (Angstrom's turbidity coefficient) by equations (2) or (3) [9]:

$$\beta = 0.55^a \left(\frac{3.912}{V_{is}} \cdot 0.01162 \right) \left(\frac{0.44(V_{is} - 5)}{18} + 1.132 \right) \quad (2)$$

Equation (2) proves to be accurate enough when the value of particle size distribution exponent a is 1.3. For the values of a different to 1.3 equation (3) proves to be more accurate.

$$\beta = \left(\frac{3.912}{V_{is}} \cdot 0.01162 \right) \{ (16.23285 + V_{is})(F - G \cdot a) + H \} \quad (3)$$

Where

Vis=Horizontal visibility [km]

$F = 2.3575E^{-02}$,

$G = 9.387E^{-03}$ and

$H = 0.278863$

Equation (2) and (3) do not cover visibilities in fog. During fog, the size of the particles becomes very big hence none of these equations are applicable in that condition. The estimation of diffuse radiation is done by using modified Pinazo model [4].

2.2 Surface radiation model

Two types of surface models are implemented here, one is for stationary surfaces and the other is for moving as well as for flying surfaces. The modeller has to define the surface orientation of each surface in the surface model parameter window. If the surface is moving, then modeller has to give the initial surface orientation in the surface model and the change in surface orientation with time in the global model. The surface model reads the change in surface orientation from the global system and calculates the solar incident angle on each surface for each time step. Figure 3 shows the parameter window of the surface radiation model

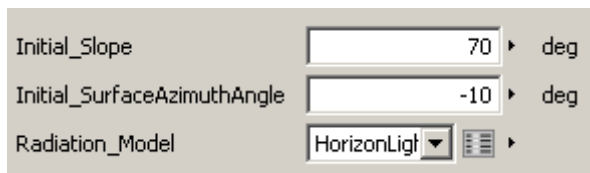


Figure 3: Surface radiation model parameter window

Four different radiation models are implemented [6]. The modeller has to select one of the following radiation models from the drop down menu,

- Isotropic model: All diffuse radiation is uniformly distributed over the sky dome.
- Circumsolar model: The effect of circumsolar radiation and horizon brightening is taken into account.
- Iso-circumsolar model: The portion of the diffuse radiation is treated as circumsolar and the remaining portion is treated as isotropic.
- Horizon brightening model: In addition to isotropic diffuse and circumsolar radiation, the Reindl model also accounts for horizon brightening.

When the model calculates clear sky radiation, the results of circumsolar model and isotropic model are

the same. All four models are implemented to use under clear sky conditions as well as under overcast conditions. The surface radiation model can further be connected to wall models and/or window models.

3 Estimated solar radiation

3.1 Stationary surface at ground

Figure 4 shows a comparison between estimated solar radiation under clear sky conditions and the measured solar radiation. The measured data shown in the figure 4 were taken from Fraunhofer IBP's weather station for the 10th of September 2011 [8].

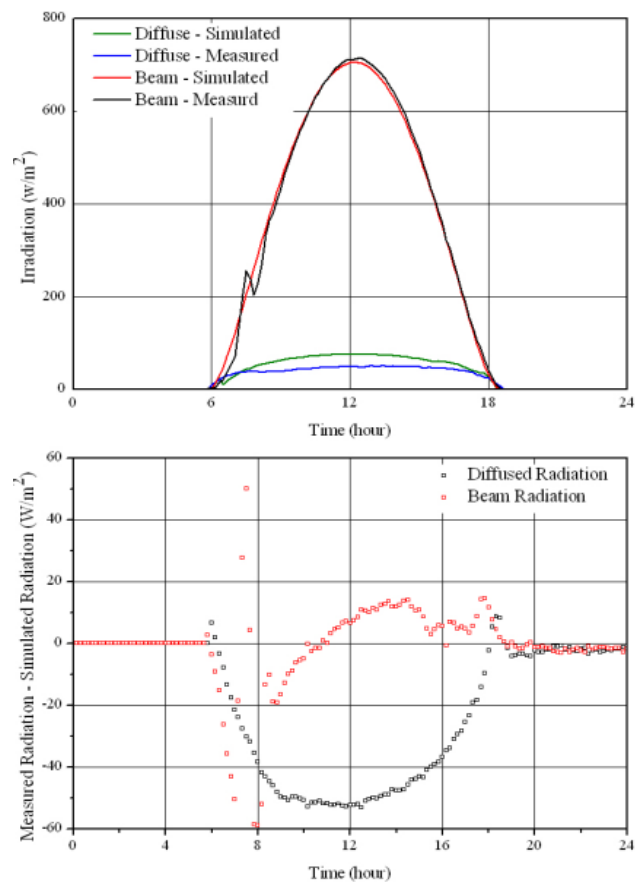


Figure 4 : Comparison of estimated radiation with measured radiation (top) and difference in measured and estimated radiation (bottom)

For almost the whole day, there were no clouds in the sky except for some time between 7 am and 9 am, where one can see a larger deviation between measured and simulated beam radiation. This difference can be reduced if the cloud factor is known. The current model can calculate overcast conditions if the cloud factor is known in advance. For the clear sky condition, the difference between simulated and measured beam radiation is less than $\pm 20 \text{ W/m}^2$.

3.2 High altitude solar radiation

At cruise altitudes, solar radiation intensity is much higher because the solar beam has to travel less distance in the atmosphere.

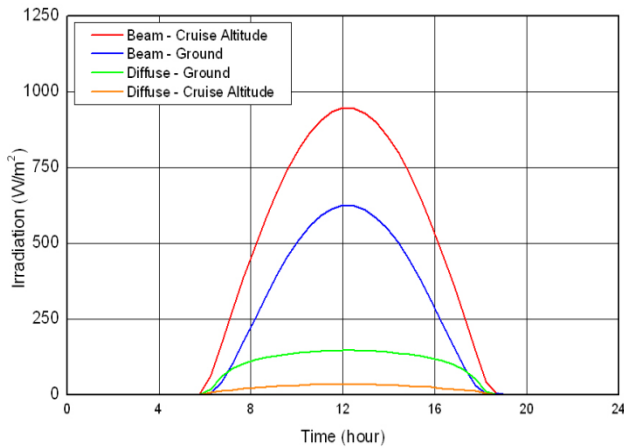


Figure 5 : Ground level and high altitude solar radiation

Figure 5 shows the difference between the solar radiation on a horizontal surface at ground level (in Holzkirchen, Fraunhofer IBP, Germany) and at cruise altitude of 35,000 ft. Results shown are for the spring time. During hot summer days, solar radiation at cruise altitude can go up to 1200 W/m².

3.3 Solar radiation on aircraft

Solar heating can contribute significantly to thermal loads of an aircraft, especially when flying at high altitudes. Solar radiation affects e.g. aircraft cockpits directly through the windshield and cabins through windows. Heat dissipated by internal heat sources and heating by direct solar radiation has an adverse effect on thermal comfort of passengers, cabin crew and pilots which requires considerable amount of cooling air in the cabin and in the cockpit. When the aircraft is on ground at some hot and humid place the effect of solar heating is significant. While the aircraft is on ground, temperature of the surfaces exposed to direct solar radiation are typically 20 K higher than ambient temperature, depending on the thermal capacity of the surface material and geographic location. Figure 6 shows the global sun position model and environment model connected with a flight profile. It also shows a wall structure model and window model connected with the surface orientation model.

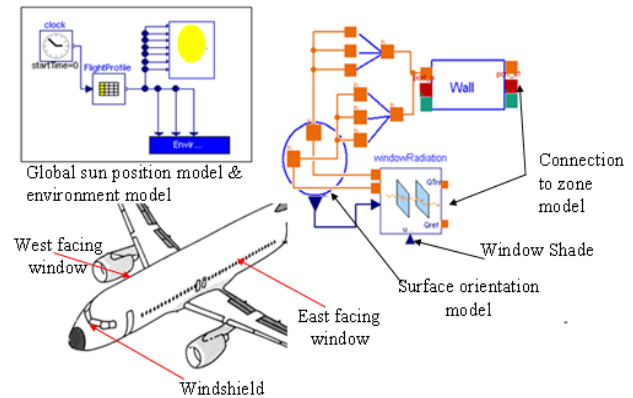


Figure 6 : Model to simulate solar radiation on an aircraft

The model shown in the figure 6 reads the flight profile (longitude, latitude, altitude, time, day of the year) and accordingly calculates the irradiation on differently oriented surfaces of the aircraft skin. The assumption of clear sky condition is fairly accurate and viable to use at cruise altitude, as there are not much clouds present at this altitude. Environmental parameters such as ambient pressure, ambient temperature, humidity, skin temperature etc. are implemented as functions depending on the flight profile.

The surface orientation model for the aircraft fuselage is a discretised cylinder model. The cylindrical surface is discretized into a number of rectangular strips where each strip has a different surface orientation and each strip is an individual surface which will calculate its new orientation as per its initial position and the given flight profile. The incident angle for each surface is different. This cylindrical surface model is then further connected to the window and wall model.

Figure 7 shows the solar irradiation on aircraft windows and windshield. The simulation is done for a flight from Munich international airport to Johannesburg international airport. There are several assumptions made such as: flight duration is 10 h 30 min, departure time from Munich is 7 am, duration of taxiing at departure airport and at arrival airport is ignored, initial take-off and approach is ignored. The simulation is done for 21-March, 21-June and 21-Dec. It is assumed that the flight takes 30 min to reach cruise altitude and 45 min for descent and initial approach. The time to reach cruise altitude depends on the several factors like type of aircraft, weight of aircraft, allowable angle of attack and angle of turn etc. The flight profile considered here cannot be applied as a standard profile; it is

purely based on close approximation. The cruise altitude considered for this simulation is 39,370 ft.

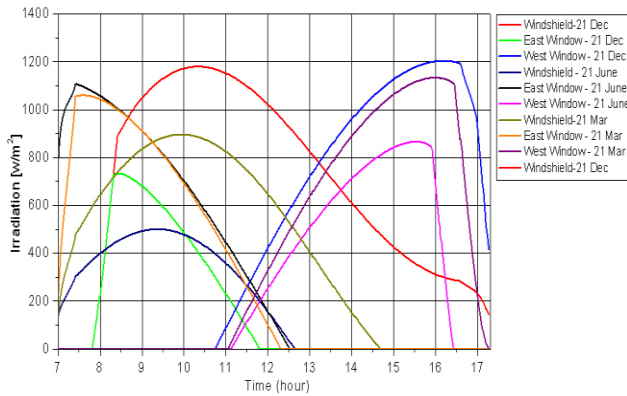


Figure 7 : Incoming solar radiation on window and windshield outer surface.

While observing the figure 7, one should keep in mind that when it is winter in the northern hemisphere, it is summer in the southern hemisphere. The three dates considered in the simulation represent the summer solstice, winter solstice and equinox. The results shown in the figure 7 can be considered as irradiance over the outer surface of windshield and window, and not as the amount of irradiance entering into cockpit and cabin. The window and windshield will absorb some of the solar radiation, some of the radiation will be reflected and the remaining will be transmitted.

4 View Factor Model

4.1 Basics

View factors between two surfaces are dependent on the geometry of the surfaces and their orientation. The view factor can be interpreted as the fraction of diffusive radiant heat exchange between surface i and surface j . The view factor between two infinitesimal surface elements dA_i and dA_j is defined by equation (4). [10][11]

$$F_{ij} = \frac{1}{A_i} \int_{A_i} \int_{A_j} \frac{\cos\theta_i \cdot \cos\theta_j}{\pi \cdot r^2} dA_i dA_j \quad (4)$$

Equation (4) is the general equation of a view factor between surface i and surface j , as shown in figure 8, where r is the distance between the centres and $\cos\theta_i$ and $\cos\theta_j$ are the directional cosines. $\cos\theta_i$ and $\cos\theta_j$ can be determined by using following equation [11].

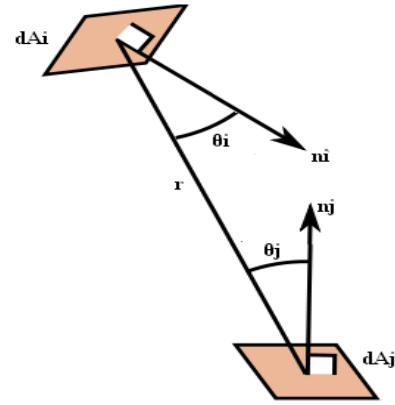


Figure 8: View factor between two infinitesimal surface elements

$$\cos\theta_i = \frac{l_i(x_j - x_i) + m_i(y_j - y_i) + n_i(z_j - z_i)}{r} \quad (5)$$

$$\cos\theta_j = \frac{l_j(x_i - x_j) + m_j(y_i - y_j) + n_j(z_i - z_j)}{r} \quad (6)$$

Where x, y, z are the coordinates of a centre of the element under consideration. When discretizing surfaces i and j with triangular elements to solve equation (4) the distance r is determined from the centres of the triangles. The areas of the elements are determined using the parallelogram theorem. The discretization yields equation (7) :

$$F_{ij} = \frac{1}{A_i} \sum \sum \frac{\cos\theta_i \cdot \cos\theta_j}{\pi \cdot r^2} \cdot dA_i dA_j \quad (7)$$

Once F_{ij} is known one can calculate F_{ji} from equation (8):

$$A_i \cdot F_{ij} = A_j \cdot F_{ji} \quad (8)$$

4.2 Modelling approach

A pre-processor is used to create a triangular surface mesh and to store it as .stl-file. This file is the input for the Modelica model described here. The model reads vertex and normal vector of each triangular facet from the .stl-file and creates both coordinate matrix and normal vector matrix. In the next step, the model will calculate the centres of each triangle, the distance r for each triangle with all the other triangles and similarly directional cosines for each triangle. The 'area function' call in the model will calculate area of each triangle. The 'view factor' function call in the model will calculate the view factor of each triangle with all the other triangles.

Finally the ‘sum view factors’ function call will give the final view factor of the whole surface with all the other surfaces. Summation of view factors is done as shown in the equation (7). All of the above functions are implemented in a view factor model. The modeller has to give only the .stl-file for each surface.

5 Application of view factor model

5.1 Closed geometry

The accuracy of the model is dependent on the meshing size. With finer meshing the accuracy of results is improved. If the meshing is coarse, the results are less accurate but the model will take less computational time.

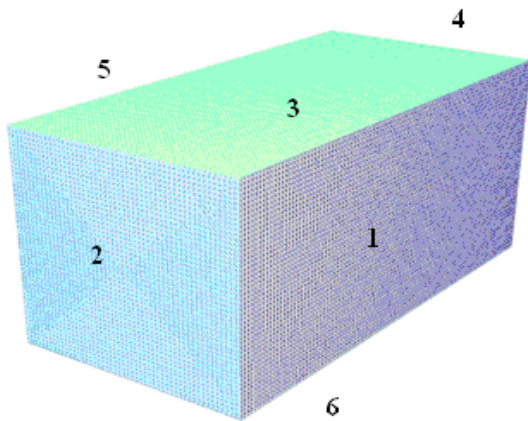


Figure 9 : Closed geometry (box)

Figure 9 shows a rectangular box with 6 surfaces. The box is 1 m long, 0.5 m high and 0.5 m wide. It is easy to solve the double area integral (DAI) for this geometry and that is the reason why such simple geometry is considered, so that the results of the Modelica model can be compared with the DAI solution. There are 6 surfaces of the box. Each surface of the box can see the other surface, so there will be 6x6 view factors but none of the surface can see itself as all the surfaces are flat surfaces hence there will be 6x6 view factors with a zeros on the diagonal of the view factor matrix. Table 1 shows the result of the Modelica model and the actual view factor values calculated by DAI and their comparison.

Table 1 : Comparison of view factors of a box meshed into 4000 triangles.

		1	e %	2	e %	3	e %
1	Modelica	0.000	0.00	0.119	2.23	0.244	1.19
	DAI	0.000		0.116		0.241	
2	Modelica	0.238	2.21	0.000	0.00	0.238	2.21
	DAI	0.233		0.000		0.233	
3	Modelica	0.244	1.19	0.119	2.23	0.000	0.00
	DAI	0.241		0.116		0.000	
4	Modelica	0.238	2.21	0.069	0.03	0.238	2.21
	DAI	0.233		0.069		0.233	
5	Modelica	0.286	0.19	0.119	2.23	0.244	1.19
	DAI	0.286		0.116		0.241	
6	Modelica	0.244	1.19	0.119	2.23	0.286	0.19
	DAI	0.241		0.116		0.286	
		4	e %	5	e %	6	e %
1	Modelica	0.119	2.23	0.286	0.19	0.244	1.19
	DAI	0.116		0.286		0.241	
2	Modelica	0.069	0.03	0.238	2.21	0.238	2.21
	DAI	0.069		0.233		0.233	
3	Modelica	0.119	2.23	0.244	1.19	0.286	0.19
	DAI	0.116		0.241		0.286	
4	Modelica	0.000	0.00	0.238	2.21	0.238	2.21
	DAI	0.000		0.233		0.233	
5	Modelica	0.119	2.23	0.000	0.00	0.244	1.19
	DAI	0.116		0.000		0.241	
6	Modelica	0.119	2.23	0.244	1.19	0.000	0.00
	DAI	0.116		0.241		0.000	

Results shown in table 1 are for a box meshed into 4000 triangles. The maximal error is 2.23 % for F_{12} , F_{32} , F_{52} , F_{62} , F_{14} , F_{34} , F_{54} and F_{64} . The minimal error is 0.03 % for F_{24} and F_{42} .

Table 2 shows the similar results as table 1 but with a bit finer meshing. The maximal error in table 2 is 0.57 % and the minimal error is 0.01 %.

Table 2 : Comparison of view factors of a box meshed into 6000 triangles.

		1	e %	2	e %	3	e %
1	Modelica	0.000	0.00	0.117	0.52	0.242	0.57
	DAI	0.000		0.116		0.241	
2	Modelica	0.234	0.49	0.000	0.00	0.234	0.49
	DAI	0.233		0.000		0.233	
3	Modelica	0.242	0.57	0.117	0.52	0.000	0.00
	DAI	0.241		0.116		0.000	
4	Modelica	0.234	0.49	0.069	0.01	0.234	0.49
	DAI	0.233		0.069		0.233	

5	Modelica	0.286	0.08	0.117	0.52	0.242	0.57
	DAI	0.286		0.116		0.241	
6	Modelica	0.242	0.57	0.117	0.52	0.286	0.08
	DAI	0.241		0.116		0.286	
		4	e %	5	e %	6	e %
1	Modelica	0.117	0.52	0.286	0.08	0.242	0.57
	DAI	0.116		0.286		0.241	
2	Modelica	0.069	0.01	0.234	0.49	0.234	0.49
	DAI	0.069		0.233		0.233	
3	Modelica	0.117	0.52	0.242	0.57	0.286	0.08
	DAI	0.116		0.241		0.286	
4	Modelica	0.000	0.00	0.234	0.49	0.234	0.49
	DAI	0.000		0.233		0.233	
5	Modelica	0.117	0.52	0.000	0.00	0.242	0.57
	DAI	0.116		0.000		0.241	
6	Modelica	0.117	0.52	0.242	0.57	0.000	0.00
	DAI	0.116		0.241		0.000	

For a box with 8000 triangles (see table 3) the error is even less. The maximal error for a box meshed into 8000 triangles is 0.086 % and the minimal error is 0.00 %. It is obvious that the error can be reduced by fine meshing but it would be interesting to see the effect of fine meshing on the computation time. Table 3 summarized results and indicates the needed computation time on a computer with – ‘Intel® Core™ i5, M 520@2.40 GHz, 2.92 GB RAM, Window 32-bit’

Table 3 : Result summary for view factor calculation

No of Triangles	Max Error [%]	Min Error [%]	Total error of closed geometry [%]	Computation time [min]
4000	2.233	0.030	1.10	34
6000	0.566	0.013	0.40	86
8000	0.086	0.000	0.05	170

The geometry under consideration is symmetric. The computation time for non-symmetric geometries can be even higher. The meshing size is the defining factor one has to define as per the level of accuracy required and computational time.

5.2 Geometry with openings

Figure 10 shows the geometry with openings on surface 1 and surface 2. The geometry shown in figure 10 is meshed into 4000 triangles. The results of a Modelica model are shown in table 4.

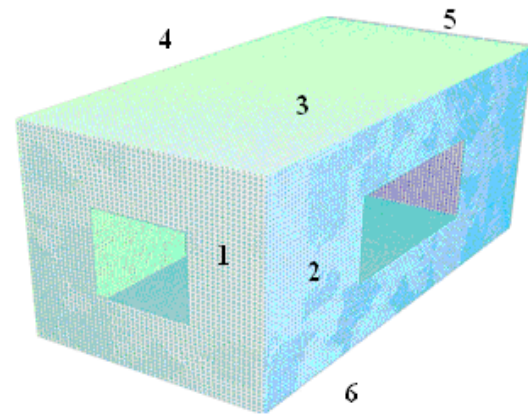


Figure 10 : Geometry with openings

The size of the rectangular box is the same as it was in the closed geometry. Therefore, only view factors concerning surface 1 and surface 2 will be different while all other results will be the same. The size of the opening on surface 1 is 0.2m x 0.2m and on surface 2 is 0.2m x 0.4m.

Table 4 : View factor matrix for the geometry with openings on surface-1 and surface-2

	1	2	3	4	5	6
1	0.000	0.215	0.240	0.240	0.068	0.240
2	0.107	0.000	0.246	0.274	0.127	0.246
3	0.101	0.207	0.000	0.244	0.119	0.286
4	0.101	0.230	0.244	0.000	0.119	0.244
5	0.057	0.213	0.238	0.238	0.000	0.238
6	0.101	0.207	0.286	0.244	0.119	0.000

6 Conclusion & Future Work

A step towards better modelling of radiative heat transfers with Modelica has been taken in the presented work. An overview of the solar irradiation modelling for stationary, moving and flying surfaces is outlined in this paper. Comparison with weather data for a clear day shows that results of estimated clear sky radiation at ground level are accurate. A further model has been developed to determine the view factor between differently oriented surfaces. Comparison with an analytical solution shows that the accuracy increases with the number of surface elements used to discretize surfaces. For the future, we intend to include a model which can calculate obstructed view factors as well. Computational time is also an area of scrutiny where we intend to investigate solutions allowing for higher speed. These developments will allow for better modelling

of radiative heat transfers when considering thermal management in stationary and mobile spaces.

7 References

- [1] L.Elterman : UV, Visible and IR Attenuation for Altitudes to 50 km, Air Force Cambridge Research Laboratories, L.G. NANSKOM FIELD, BEDFORD, MASSACHUSETTS, 1968.
- [2] John R. Howell.: A Catalog of Radiation Heat Transfer Configuration Factors, 3rd Edition, Department of Mechanical Engineering, The University of Texas at Austin, Visited on 16.March 2012, <http://www.engr.uky.edu/rtl/Catalog/>
- [3] J.M. Pinazo, J.Canada, J.V.Bosca.: A New Method to Determine Angstrom's Turbidity Coefficient: Its Application for Valencia, Solar Energy, Vol. 54, No. 4, pp. 219-226, 1995
- [4] A.Q. Malik.: A modified method of estimating Angstrom's turbidity coefficient for solar radiation models, Renewable Energy 21 (2000) 537-552.
- [5] K.Yang,G.W.Huang and N.Tamai.: A hybrid model for estimating global solar radiation, Solar Energy Vol. 70, No. 1, pp. 13–22, 2001
- [6] Soteris A. Kalogirou : Slolar Energy Engineering Processes and Systems, Elsevier Inc, 2009.
- [7] John A.Duffie, William A.Beckman.: Solar Engineering of Thermal Processes, John Wiley & Sons. Inc. 1980.
- [8] Fraunhofer IBP Weather Station Website, consulted on 06.March 2012, <http://www.ibp.fraunhofer.de/wetter>
- [9] Meinrand A. Mächler: Parameterization of solar irradiation under clear skies, Diploma Maschineningenier, Eidg.Techn. Hochschule Zürich, 1977.
- [10] Michael F. Modest, Radiative Heat Transfer – Second Edition, Academic Press, Elsevier Science, 2003.
- [11] Lilian Dobrowolski de Carvalho Augusto, Bruno Giacomet and Nathan Mendes.: Numerical Method for Calculating View Factor Between Two Surfaces, Proceedings: Building Simulation 2007

VEPZO – Velocity Propagating Zonal Model for the prediction of air-flow pattern and temperature distribution in enclosed spaces

Victor Norrefeldt Gunnar Grün
Fraunhofer-Institute for Building Physics
Fraunhoferstr. 10, 83626 Valley, Germany
victor.norrefeldt@ibp.fraunhofer.de

Abstract

This paper presents the VEPZO-model (VELOCITY Propagating ZONal model), the first three dimensional airflow model for indoor spaces that has been implemented in Modelica. The model predicts airflow and temperature distribution in a room. The main feature of the VEPZO model is that each zone has a characteristic velocity depending on entering and leaving airflows. This characteristic velocity is propagated into space ensuring the propagation of driving airflows. The VEPZO model can be interfaced to other models of the Modelica Standard library. In an application example a displacement ventilation in a twin-aisle aircraft cabin is investigated. The temperature in the occupied zones is predicted between 20.6 and 23.0 °C.

Keywords: zonal model, airflow modeling, Modelica

1 Introduction

Accurate energetic modeling of buildings and vehicles requires the consideration of included air. In a multizone model the air in a room or zone is considered perfectly mixed. Air exchanges occur between rooms or with the environment. This approach is implemented in the “Buildings” library [1]. If a higher level of detail is needed, the zonal model is an alternative that can be implemented in Modelica. A zonal model divides a room into typically 10 to 100 zones exchanging air through flow paths. At the last Modelica conference Bonvini and Leva [2] presented an implementation of a two-dimensional zonal model. In parallel to their work another Modelica-based zonal model, VEPZO (Velocity Propagating Zonal Model) has been built at the Fraunhofer-Institute for Building Physics. Former zonal models have some drawbacks that make them unsuitable for the use in Modelica. The airflow is supposed to rest in the zones and to move in the flow models. Once air enters a zone, its velocity is dissipated. Therefore, the

zonal formulation is not valid in areas with driving flows due to jets or plumes [3]. Here, Inard et al. [4] suggest using jet-, plume- or thermal boundary layer correlations to compute the amount of air entrained from the surrounding “normal” zones. A Modelica implementation of this suggestion would require the model to change its set of equations during runtime to be able to switch from the zonal model to a correlation model where needed. However, this feature is currently not provided by Modelica. Furthermore, the flow models are based on the Bernoulli-equation resulting in a square root function of the pressure difference. At zero pressure difference, the square root is numerically unstable due to its infinite gradient. Therefore, the following modifications have been made in the VEPZO model:

- Calculation of the acceleration or deceleration of the airflow between two adjacent zones with a viscous loss model
- Introduction of the length of an airflow path into the zonal equations
- Use of the airflow velocity as a property in the zones

Bonvini and Leva [2] use a similar approach, however the VEPZO model differs from their implementation in the following points:

- Use of Modelica.Media-models for air properties computation
- Use of stream-connectors
- A flow model connects to two zones only and not to adjacent flow models
- Three dimensions implemented
- Viscosity is a parameter with value 0.001

2 Implementation of the VEPZO model

The two main components of the VEPZO model are a zone model and a flow model (Figure 2). The zone

(cube) and the flow (grey rectangle) models are connected by ports (rhombs) to form a room. These ports allow the exchange of relevant information between the flow and the zone model:

```

replaceable package Medium = Modelica.Media.Air.SimpleAir
Modelica.SIunits.Pressure p;
Records.Position position;
Records.Velocities velocities;
flow Modelica.SIunits.MassFlowRate mdot;
stream Modelica.SIunits.SpecificEnthalpy h;
stream Modelica.SIunits.Density d;
stream Modelica.SIunits.MassFraction Xi[Medium.nXi];
stream Real ExtraProperty[Medium.nC];
Real dv_perp[2];
Real sum_d2v_perp_weighted;
    
```

Figure 1: Implementation of connector, position and velocities contain coordinates and characteristic velocity of zone, dv_perp is the gradient of characteristic velocity perpendicular to flow direction, $sum_d2v_perp_weighted$ is a quantity used for viscosity computation

The flow models have two ports to connect adjacent zones. Each zone has six ports, one for each boundary. A Boolean parameter is assigned to each port to make the distinction whether the port is connected to a flow model or whether it is adjacent to a room boundary surface. Furthermore, each zone has a heat port (red square) allowing heat exchanges with models of other components like e.g. heat sources or walls. Air properties are computed from Modelica.Media. Depending on the application, the air model can be changed from dry to moist air. Even pollutants could be taken into account.

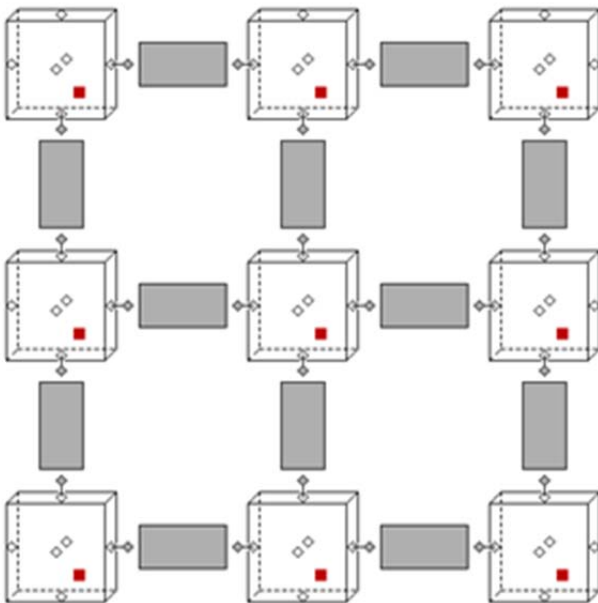


Figure 2: VEPZO model in x-z direction (y not shown); cubes: zones; grey rectangles: flows; rhombs: airflow ports; red solid squares: heat ports.

The main task of the zone model is to compute the mass and enthalpy balance and air properties (density, enthalpy, pressure, temperature, etc) using air models of Modelica.Media. Furthermore it determines a characteristic velocity and viscous losses. The main task of the flow model is to compute the airflow rate between two adjacent zones. Furthermore, the flow models are used to calculate the velocity gradient needed for the calculation of viscous losses. The governing equations will be more precisely described in the following sections.

2.1 Zone model

The mass and energy conservation are implemented in the VEPZO model in the same way as in former zonal models. Air contained in a zone i of volume V_i with density ρ_i and specific enthalpy h_i is assumed to be perfectly mixed. In the zones, the dynamic conservation of mass (Equation (1)) and enthalpy (Equation (2)) are implemented. The mass conservation takes into account the amount of air $m_{i,j}$ exchanged with adjacent zones and airflows provided by various sources or sinks $m_{source,i}$ (ventilation, openings, etc.) in zone i . When steady state is reached the sum of all exchanged airflows in a zone becomes zero. Mass flows are defined as flow variable and enthalpies as streams. This ensures the proper sign attribution to flows an enthalpy selection depending on flow direction. Heat flows Q due to convection to walls or heat sources contained in the zone are added to the thermal energy balance.

$$\frac{\partial \rho_i}{\partial t} = \frac{\sum_j \dot{m}_{i,j} + \sum_{sources} \dot{m}_{source,i}}{V_i} \quad (1)$$

$$V_i \cdot \rho_i \cdot \frac{\partial h_i}{\partial t} = \sum_j \dot{m}_{i,j} \cdot h_{i/j} + \sum_{sources} \dot{m}_{source} \cdot h_{i/source} + \sum_{heat\ flows} \dot{Q} \quad (2)$$

A new feature of the VEPZO model is that a characteristic velocity vector (u,v,w) is assigned to the zones. Knowing the mass flow and its direction across each of the zone's surfaces, the flow velocity across these surfaces is determined: u_{left} , u_{right} , v_{front} , v_{back} , w_{bottom} , w_{top} for the left, right, front, back, bottom and top surfaces. If a zone is adjacent to a wall, the mass flow and velocity across the corresponding

surface are considered to be zero. For each Cartesian direction the flow pattern is checked. If air flows through the zone, the entering velocity is assigned to the characteristic velocity component. If air enters from both sides, the difference of the velocities is assigned. If air leaves on both sides, zero is assigned (shown for x-direction in Figure 3 and Table 1). Iterating this procedure for all Cartesian directions yields the characteristic velocity of a zone.

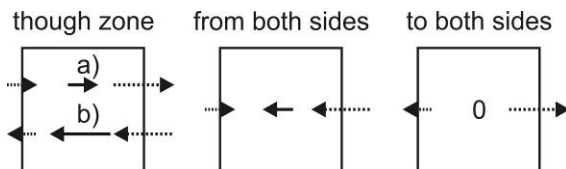


Figure 3: Assignment of characteristic velocity to a zone, dotted arrows: airflow across zone boundaries, solid arrow: characteristic velocity of zone. Left: flow through the zone, case a) left to right, case b) right to left, Middle: Air enters zone from both sides, Right: Air leaves zone on both sides

Table 1: Assignment of characteristic velocity components of a zone

Flow through zone	Characteristic velocity
left → right	u_{left}
right → left	u_{right}
left and right → zone	$u_{\text{left}} - u_{\text{right}}$
zone → left and right	0

The zone shares the information about its characteristic velocity with the flow models surrounding it. This enables the VEPZO model to propagate the air-flow velocity throughout the room without needing special correlations like jets or plumes.

2.2 Flow model

Two adjacent zones are connected by a flow model computing the exchange of air between them. The VEPZO model uses flow models in x-, y- and z-directions. The assumption of the VEPZO model is that air only flows along these specific directions. A new feature of the flow model used in the VEPZO model is that the length of a flow path is taken into account.

The air densities ρ considered in the flow models are the average densities of the air contained in the adjacent zones. The use of the actualStream-notation for

the density showed to result in longer simulation times. Furthermore, density differences in indoor applications are not considered important enough to introduce a major error to the simulation when averaging.

2.2.1 Geometrical properties of the flow model

Figure 4 shows the definition of zones and flows in the z-direction. Two zones i, j of height $\Delta z_i, \Delta z_j$ and equal ground area $A = \Delta x \cdot \Delta y$ are connected by the flow model “Flow_{ij}”. The flow model is of area A and height Δz_{ij} . This height is equal to the distance between centres of zone i and j . The definition of flow models in the x- and y-direction is analogous.

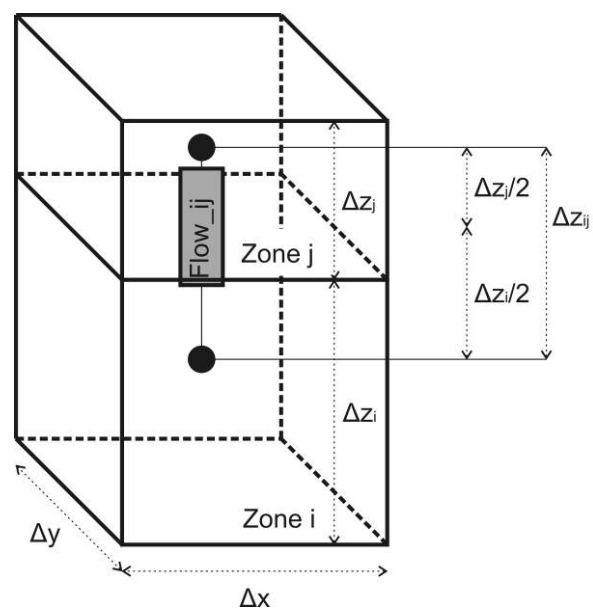


Figure 4: Definition of zones and flows in z-direction.

2.2.2 Forces acting on flow path

The flow model computes the airflow acceleration or deceleration from the forces acting on it. These forces are briefly described.

Pressure difference

Air contained in each zone has a certain pressure. When two zones of common surface A are connected by a flow model they process their pressure information p_i and p_j . The flow model calculates the resulting force F_P (Equation (3)).

$$F_P = -A \cdot (p_j - p_i) \quad (3)$$

Momentum difference

The characteristic velocity vectors of adjacent zones are processed to the flow model. According to the flow direction (x, y or z) the flow model chooses the proper component of the velocity vectors (u_i, v_i, w_i and u_j, v_j, w_j) to compute the force F_M resulting from the momentum difference between the adjacent zones (Equation (4) for x-direction).

$$F_{M,x} = -\rho \cdot A \cdot (u_j^2 - u_i^2) \quad (4)$$

Gravitational forces

Gravitational force F_G only occurs in the z-direction. For the x- and y-direction, this force is zero. To compute the gravitational force, the area and length of the flow path are considered according to Figure 4.

$$F_G = -\rho \cdot g \cdot A \cdot \Delta z_{ij} \quad (5)$$

Viscous forces

The viscous forces act parallel to the flow direction. In the selected approach of the VEPZO model, flows are connected and exchange information with zones only. However, to calculate the shear stress, an information exchange between parallel flow models would be necessary. To avoid connections between the flow models, viscous losses are calculated in the zone models but used in the flow models.

The characteristic velocity vector provided by zones enables the flow model to calculate the gradient of the two velocity components perpendicular to the flow model direction. For example, a flow model in z-direction can deliver the variation of the characteristic velocities u_i, u_j and v_i, v_j along the height Δz_{ij} (Figure 5). If a wall is adjacent to the zone, the velocity at the wall is assumed to be zero. Therefore, the gradient is equal to the characteristic velocity divided by half the distance of the zone's centre from the wall. Equation (6) provides an overview of gradient calculations in x-direction.

Gradient in flow model		Gradients at walls	
$\frac{\Delta v_{ij}}{\Delta x_{ij}}$	$\frac{\Delta w_{ij}}{\Delta x_{ij}}$	$\frac{2 \cdot v_i}{\Delta x_i}$	$\frac{2 \cdot w_i}{\Delta x_i}$

$$(6)$$

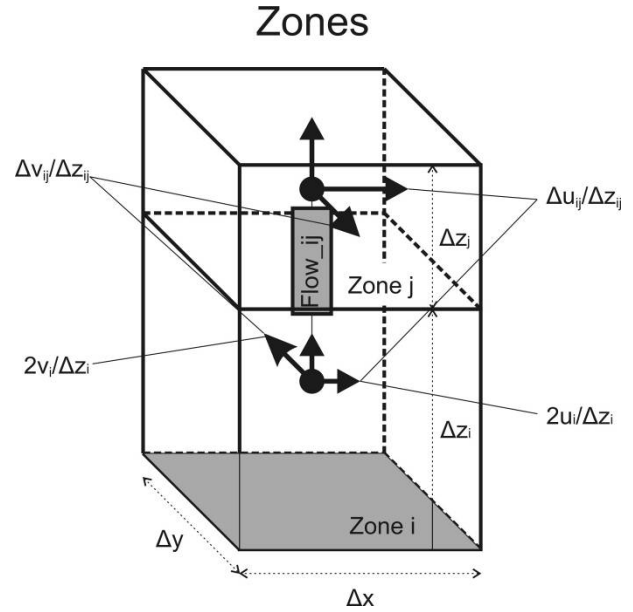


Figure 5: Computation of velocity gradient if lower wall of zone i is a wall

The gradient information is transmitted from the flow model to the zone model. In the zone model this gradient causes shear stresses on its boundaries. Summing these shear stresses along the boundaries yields the viscous forces $F_{V,x}$, $F_{V,y}$, and $F_{V,z}$ in the zones (μ : dynamic viscosity of air):

$$F_{Vx,Zone} = \mu \cdot \left\{ \left[\left(\frac{\Delta u}{\Delta y} \right)_{back} - \left(\frac{\Delta u}{\Delta y} \right)_{front} \right] \cdot \Delta x \cdot \Delta z + \left[\left(\frac{\Delta u}{\Delta z} \right)_{up} - \left(\frac{\Delta u}{\Delta z} \right)_{down} \right] \cdot \Delta x \cdot \Delta y \right\} \quad (7)$$

Because the flow model covers half of the length of each adjacent zone (Figure 4) the resulting viscous force in the flow model is the sum of half of the viscous forces in the zones.

$$F_{Vx,Flow} = \frac{1}{2} \cdot (F_{Vx,Zone,i} + F_{Vx,Zone,j}) \quad (8)$$

2.2.3 Computation of airflow between zones with viscous loss model

The forces acting on a flow path are summed up. This yields the acceleration of the portion of air contained in the flow path connecting two zones.

$$A \cdot \Delta x_{ij} \cdot \rho \cdot \frac{du}{dt} = F_p + F_{M,x} + F_{Vx,Flow} \quad (9)$$

By this procedure, the distance between two zones and the area of the flow path are introduced into the model. The total loss along a flow path therefore becomes independent from the number of zones. Furthermore, no square root function is needed and numerical problems due to an infinite derivative do not occur.

The mass flow is obtained straightforward from the velocity in a flow path. This mass flow information is then transmitted to the zone model.

$$\dot{m}_x = \rho \cdot A \cdot u \quad (10)$$

2.2.4 Modifications for non-cubic boundary zones

The idea of zonal modelling is to decompose a space into rectangular elements. However, if the modelled space is non-rectangular, non-cubic zone elements might be needed at boundaries. For this, a zone type similar to the standard zone type described in the previous sections is introduced where the sizes of each of the six boundaries surfaces, the zone's volume, its centre of gravity and its characteristic lengths can be entered manually to better match the actual geometry. For other geometries such as triangles, not needed sides of the element are attributed a very small size slightly above zero.

2.2.5 Estimation of model coefficients

In the viscous loss model, the viscosity is used as a parameter to tune the model. Similar to the idea of using a turbulent viscosity to take into account losses, an apparent viscosity is used instead of the dynamic viscosity. During implementation of the VEPZO model, $\mu = 0.001 \text{ Pa}\cdot\text{s}$ produced results that are in good accordance at steady state with case studies presented in previous publications [5-7]. Transient results have not been validated yet.

3 Investigation of a novel aircraft cabin ventilation

In this application example of the VEPZO model a novel ventilation system for an aircraft cabin will be investigated.

In current cabin designs air is supplied by ceiling inlets and extracted by slots in the dado-area on the left and right side of the cabin (Figure 6). This ventilation design leads to mixing ventilation.

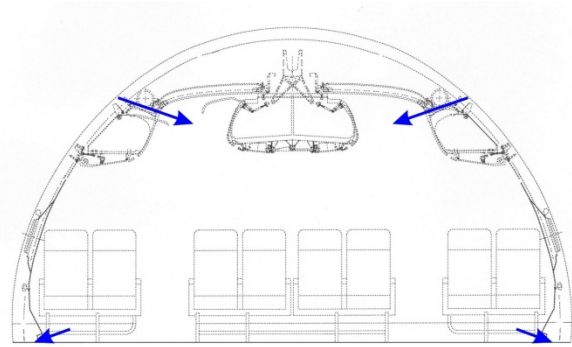


Figure 6: Cabin mixing ventilation air supply and extraction

In this paper a displacement ventilation is investigated where air is supplied by the aisle (60%), dado (20%) and side (20%) inlets. The total airflow rate is 0.5 kg/s .

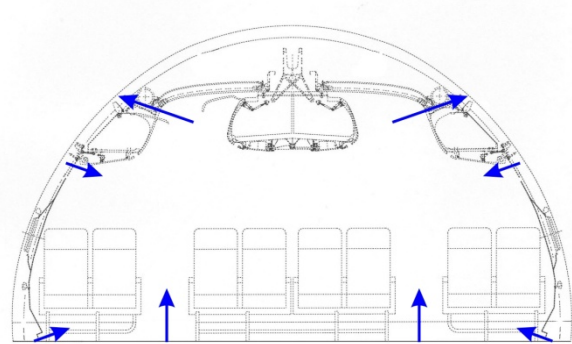


Figure 7: Cabin displacement ventilation air supply and extraction

3.1.1 Aircraft cabin ventilation – Model

The implemented aircraft cabin model is derived from the Airbus A310 mock-up placed in the Fraunhofer Flight Test Facility [8] and represents a 9.3 m long and 5.3 m wide cross-section of the fuselage including the crown area, stowage bins, cabin, left and right triangle areas, cargo compartment and bilge (Figure 8). The air volume of the cabin is modelled by the VEPZO model, the air volumes of other compartments are modelled by a thermal capacitor with the appropriate thermal mass.

The cabin is subdivided into 5×3 zones in the lower, occupied zone and into one zone for the upper part. In depth, the cabin is divided into three zones. The air supplies at the side, aisle and dado openings are modelled as mass flow sources. The ceiling outlet is modelled as a fixed pressure of 750 hPa corresponding to cabin pressure at flight altitude. The tempera-

ture of supplied air flows is controlled by the temperature in the adjacent zones (side and dado) or the average temperature in the occupied zone (aisle) (Table 2).

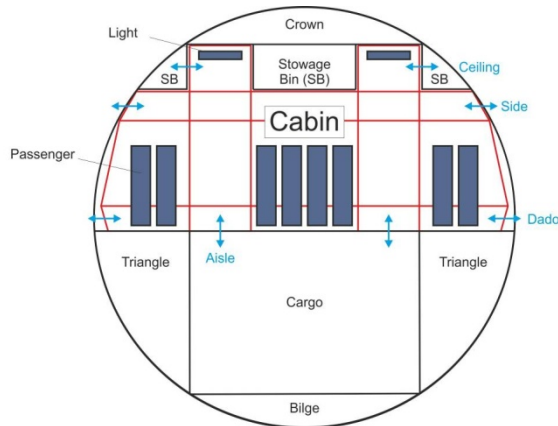


Figure 8: Fuselage section: blue arrows: inlets/outlets, red lines: zone limits in VEPZO model, black lines: walls

Table 2: Temperature setpoints and control locations

Inlet	Contol by	Setpoint
Side	Adjacent zone	22 °C
Dado	Adjacent zone	21 °C
Aisle	Average in occupied zone	22 °C

Walls are modelled by a succession of heat capacitances and heat resistances from the Modelica Thermal library. Three materials (Table 3) are used in five wall layouts (Table 4) to model fuselage enclosures. Walls in the cabin are further decomposed into facets according to the size of the adjacent zones. These facets exchange convective heat with the air volumes. For the radiation model, surface temperatures of the facets of one wall are averaged. Radiation is calculated between these averaged wall surfaces using a radiation model suggested by Wetter et al. [9]. In other compartments, walls are not further decomposed into facets but exchange convective heat with the air node and radiative heat with other walls in the compartment.

Further admitted parameters (heat source intensities, convective heat transfer coefficients, long wave emissivity of surfaces, outside air temperature) are shown in Table 5.

Table 3: Material Parameters

Material	Density (kg/m ³)	Specific Heat Capacity (J/kg·K)	Thermal Conductivity (W/m·K)
Aluminium	2700	835	235
Lining	1000	1500	0.16
Polyimide	1.2	1006	0.04

Table 4: Wall models

Name	Layers	Used for
Aluminium	Aluminium: 3 mm	Outside Wall Bilge Cabin Floor Cargo Floor
Thin Lining	Lining: 3 mm	Stowage bin
Thick Lining	Lining: 10 mm	Walls Car-go/Triangle
Cabin Outside Wall	Aluminium: 3 mm Polyimide: 80 mm Lining: 5 mm	Outside Walls Cabin
Other Outside Wall	Aluminium: 3 mm Polyimide: 80 mm	Outside Walls Crown and Triangle

Table 5: Heat flow related parameters

Parameter	Value
Convective Heat Transfer Coefficient	5 W/m ² ·K
Long Wave Emissivity of surfaces	0,95
Outside Air Temperature	-27 C
Heat dissipation by passengers	
...by radiation	72 x 37,5 W
...by convection	72 x 37,5 W
(...total)	(5400 W)
Heat dissipation by lights	
...by radiation	1000 W
...by convection	1000 W
(...total)	(2000 W)

3.1.2 Aircraft Cabin Ventilation – Results

The simulation takes 31.9 s to converge on an Intel® Core™ i5 CPU @ 2.35 GHz 2.98 GB Ram computer. The simulation time is set to 10000 s as steady-state is achieved by then. “Radau IIa – order 5 stiff” is used to solve the equations. Figure 9 shows the

resulting supply and exhaust temperatures, temperatures in the zones and in the other compartments.

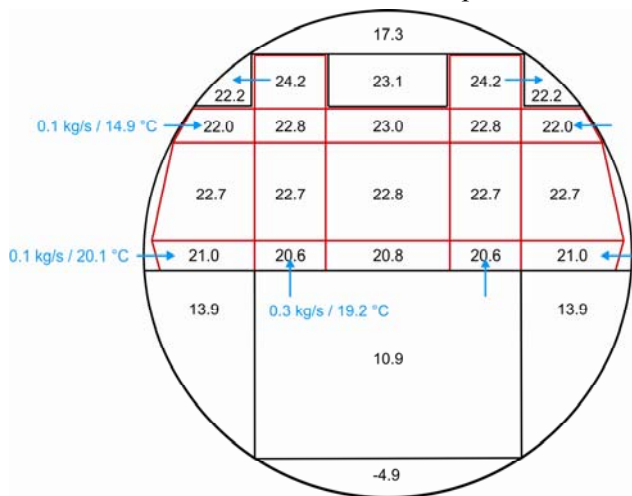


Figure 9: Results for cabin displacement ventilation system (Temperatures of zones in °C)

The warmest point in the occupied zone is just below the middle overhead bin. Here, air supply openings providing fresh air are relatively far away but heat production by passengers is relatively high. In spite of these adverse effects, the temperature is still in a comfortable range. Under the left and right overhead bins the lateral air supply avoids that the temperatures further raises compared to the zone below.

4 Conclusion

This paper presents the VEPZO model, a zonal model implemented in Modelica. The model makes use of and can be interfaced to models of the Modelica-Standard libraries. Compared to former zonal models, the VEPZO model is better suited for use in the Modelica environment.

In the shown application example the VEPZO model is used to estimate the impact of a displacement ventilation system in a two-aisle aircraft cabin. The simulation time is acceptable.

The use of Modelica to solve this problem showed to be advantageous as many of the auxiliary components (walls, air in other compartments, air properties in zones) are modeled with predefined models allowing the research engineer to concentrate on the core of the development, in this case the VEPZO model.

5 Acknowledgements

This research work has been conducted with financial support from the German Bundesministerium für Bildung und Forschung (support code 20Y0907E). The author is responsible for the content of this publication.

6 References

- [1] Wetter, M., Zuo, W., Nouidui, T.: Recent developments in the Modelica "Buildings" Library for Building Energy and Control Systems, 8th International Modelica Conference, 20.-22. march 2011, Dresden, Germany
- [2] Bonvini, M., Leva, A.: Object-oriented sub-zonal room models for energy-related building simulation, 8th International Modelica Conference, 20.-22. march 2011, Dresden, Germany
- [3] Wurtz, E., Nataf, J. M., Winkelmann, F.: Two- and three-dimensional natural and mixed convection simulation using modular zonal models in buildings, International Journal of Heat and Mass Transfer, Volume 42, pp. 923-940, 1999
- [4] Inard, C., Bouia, H., Dalicieux, P.: Prediction of air temperature distribution in buildings with a zonal model, Energy and Buildings, Volume 24, pp. 125-132, 1996
- [5] Norrefeldt, V., Nouidui, T., van Treeck, C. et al.: Erstellung eines isothermen, zonalen Modells mit Impulserhaltung, BauSIM 2010, 22.-24. september 2010, Vienna, Austria
- [6] Norrefeldt, V., Grün, G.: Validation of the velocity propagating zonal model – VEPZO, Roomvent 2011, 19.-22. june 2011, Trondheim, Norway
- [7] Norrefeldt, V., Grün, G., Sedlbauer, K.: VEPZO - Velocity propagating zonal model for the estimation of the airflow pattern and temperature distribution in a confined space, Building and Environment, Volume 48, pp. 183-194, 2012
- [8] Fraunhofer IBP: Indoor Climate Systems Flyer, 2012
- [9] Wetter, M., Zuo, W., Nouidui, T.: Modeling of heat transfer in rooms in the Modelica "Buildings" Library, Building Simulation 2011, 14.-16. november 2011, Sydney, Australia

Modeling and Testing of the Hydro-Mechanical Synchronization System for a Double Clutch Transmission

Hua Huang Sebastian Nowoisky René Knoblich Clemens Gühmann
Technische Universität Berlin

Chair of Electronic Measurement and Diagnostic Technology
Sekt. EN 13, Einsteinufer 17, 10587 Berlin

{hua.huang@campus., sebastian.nowoisky@, r.knoblich@, clemens.guehmann@}tu-berlin.de

Abstract

Synchronization is a core component in the automotive powertrain. It uses friction and locking elements to synchronize the occurring speed difference during gear shifting. The optimization of this shifting process is of high interest in respect to fuel consumption and comfort considerations. Moreover, for the model-based calibration of automated transmissions, detailed simulation models of the synchronization system are also necessary. Highly accurate models allow simulation of nonlinear effects having a major influence on the shifting process. Currently, with less detailed models only rough estimations of the shifting process are possible, it has a reduced meaning for the precise calibration.

This paper uses a popular double clutch transmission (DCT) as the research object and presents the detailed hydro-mechanical synchronization model. Firstly, an introduction to the theory of the synchronization is given. Subsequently, a Modelica[®] based synchronization model consisting of hydro-mechanic actuators and gear shifting synchronizers is presented. Finally, these modules are discussed in detail and evaluated based on different simulations. A comparison with measurement data from a test bench is also included in the end.

Keywords: synchronization; hydraulic; gear shifting; double clutch transmission; physical modeling; automotive

1 Introduction

Due to the location of the synchronization in the automotive powertrain, this system has a crucial influ-

ence on the shifting quality. The shifting quality can be judged by:

- the duration of the shifting process
- the changes of vehicle longitudinal acceleration during shifting (shifting jerk)
- the oscillation to the powertrain
- the acoustic phenomena like shifting or impact noise

With conventional, less detailed models of the synchronization containing simple clutch elements as synchronization [1, 2], only three stages of the synchronization process is modeled:

- neutral position
- friction phase (synchronization)
- engaged position

In this paper a more complex simulation model of the synchronization is derived to describe certain detailed nonlinear phenomena during shifting (see section 2). Such a detailed modeling of synchronization is necessary for the model based calibration. The purpose of this calibration process is the adaption of control parameters to improve the shift quality between successive shifts. Furthermore an in-depth model provides the user with a fundamental understanding of the components composition principle and the system working function.

A 7-speed DCT with dry clutches is used here as the research object. For this transmission, a dynamic simulation model of the hydro-mechanical synchronization system is derived. This model could be used for the function development within the V-development process [3].

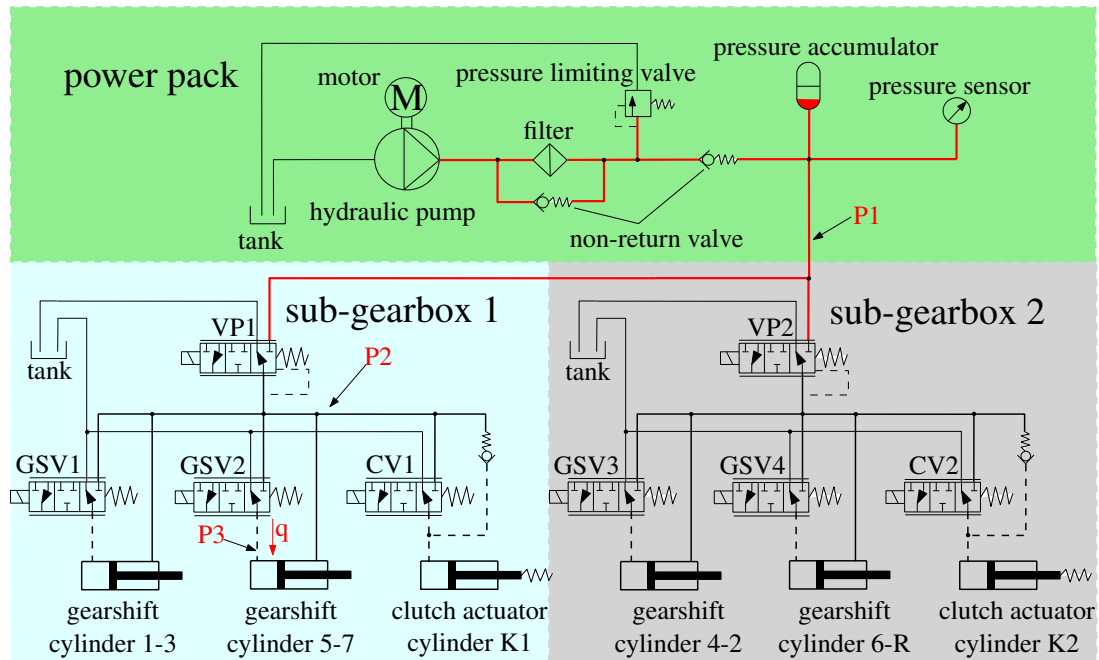


Fig. 1: Hydraulic system plan [4, 5]

In section 2 the basic components of the hydro-mechanical actuators are introduced and the synchronization process is described in detail. Then section 3 presents the simulation results of the physical model. The test bench measurements from an AMT with similar synchronization components are also compared. Finally, a summary and further research objectives are concluded.

2 Modeling

The whole synchronization system is divided into 2 parts: hydraulic and mechanical components. The hydraulic components are mainly supplying required oil pressure and flow while the remaining components are used to perform the mechanical actuator behavior and the synchronization process.

2.1 Hydraulic Components

The hydraulic subsystem consists of:

- a hydraulic pump
- magnetic valves
- gearshift cylinders

Hydraulic fluid is pumped from the tank to the pressure accumulator where it is stored under high pressure. The pump is controlled by a bang-bang controller which guarantees a pressure level between 40 and 60 bars [4]. When the oil circuit has got

enough power to drive the gearshift cylinders, the magnetic valves will control pressure and flow of relevant branches.

There are mainly two types of magnetic valves included: pressure-control valves and flow-volume valves. The pressure-control valves are used to supply the corresponding sub-gearboxes under constant pressure levels. The flow-volume valves are used to control the movement of the gearshift and clutch actuator cylinders. The hydraulic plan is depicted in Figure 1, in which each flow-volume valve controls the left chamber of a gearshift cylinder while its right chamber is controlled directly by a pressure-control valve.

2.2 Mechanical Components

2.2.1 Synchronizer and Actuation Module

Synchronizers reduce speed difference through friction and locking elements during the gear shifting process. In this paper, a widely used single-taper synchronizer based on the "Borg-Warner" system (refer to [6]), shown in Figure 2, is used as a detailed example for the synchronization process.

The components of the synchronization are named (compare [7]):

- ① idler gears with needle bearings
- ② synchronizer hub with selector teeth and friction

- taper
- ③ synchronizer ring with counter-taper and locking tothing
 - ④ synchronizer body
 - ⑤ gearshift sleeve
 - ⑥ transmission shaft

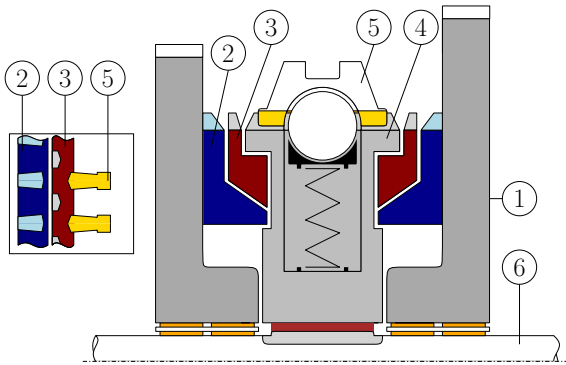


Fig. 2: Draft of the synchronization [6, 7]

During the synchronizing process, the selector fork supplies the gearshift force F_S for synchronization as the resultant of 4 forces exerted upon it: Shifting force F_C from the hydraulic cylinder, locking force F_{lml} from the detent pin, bearing friction F_{fl} , and acceleration force F_{al} , as expressed in Equation 1. The mechanic diagram of the shift actuator is presented in Figure 3.

$$F_S = F_C - F_{lml} - F_{fl} - F_{al} \quad (1)$$

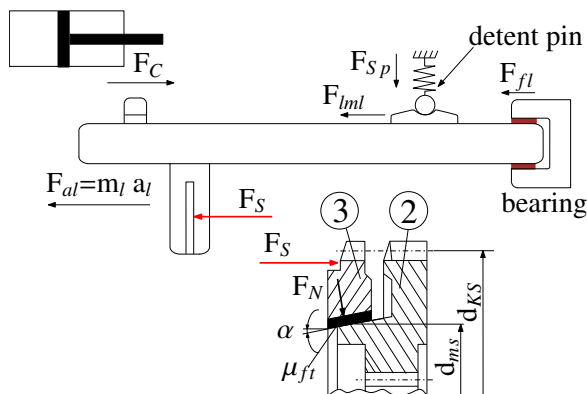


Fig. 3: Force diagram of shift actuator [7]

The detent pin showed in Figure 3 is designed to support the gearshift movement and guarantee determined positions. During the gearshift process from the neutral position to a shifted position, the detent pin introduces a counter force to the movement of the selector fork at the beginning and accelerates the fork after synchronization. This force characteristic can be

calculated by Equation 2 and is depicted in Figure 4. The locking force depends on the spring force F_{Sp} , the ramp angle γ relative to initial basis and the friction angle δ_F acting against the movement direction [7, 8]

$$F_{lml} = F_{Sp} \tan(\gamma + \delta_F) \quad (2)$$

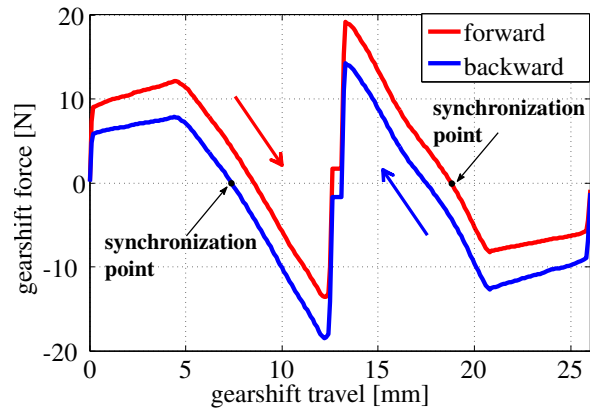


Fig. 4: Contour of ramp profile

The gearshifting process can be divided into five stages according to the gearshift position, speed difference, actuation forces and torques [6]. This classification is based on the assumption that at the beginning the gearshift sleeve is in the neutral position (see Figures 2, 3 and 5):

Stage 1: Gearshift force F_S causes an axial movement of the gearshift sleeve (5) and triggers the gearshifting process. The movement stops when the synchronizer ring blocks the gearshift sleeve.

Stage 2: The axial force is transmitted from the gearshift sleeve to the synchronizer ring (3), resulting in a friction torque T_R which is much larger than the gearing torque T_Z . At this stage the speed difference between the idler gear and transmission shaft will be reduced to zero.

Stage 3: When the speed difference is close to zero, the friction torque T_R vanishes. At this moment the synchronizer ring turns back to release the gearshift sleeve.

Stage 4: The gearshift sleeve begins to move until it encounters the synchronizer hub's (2) external gearing. Speed difference increases again as the synchronizing torque diminishes.

Stage 5: The whole synchronization process is completed as soon as the gearshift sleeve tothing engages the synchronizer hub's gearing. The power flow is transmitted from the transmission shaft (6) to

the gear ①.

Figure 5 shows the synchronization process with locking of the synchronizer ring and synchronizer hub.

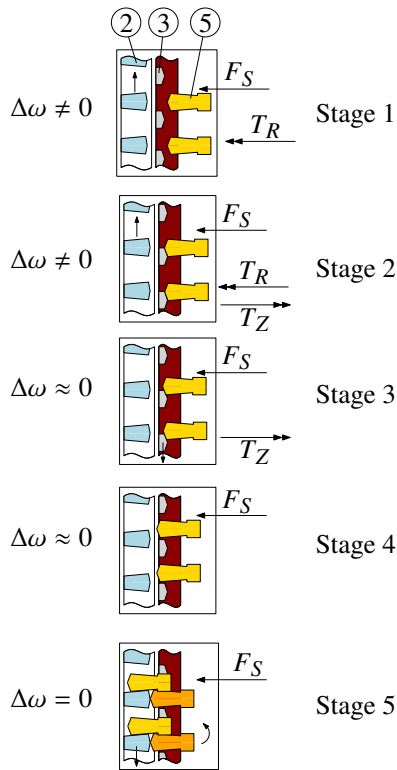


Fig. 5: Synchronizing process

2.2.2 Status Determination Module

This module is created based on Modelica[®], it uses these 3 factors as mentioned above: the gearshifting position, difference speeds, actuation force and torque, to determine the synchronization process (Figure 6 shows the flowchart of status determination). The appropriate calculations of the friction torque T_R and gearing torque T_Z are also realized here.

The detailed torque values are changed according to the synchronization stages: The friction torque T_R , given by Equation 3 (applied to stages 1 and 2), is calculated through the gearshift force F_S , the number of friction surfaces j and some other geometric values. The gearing torque T_Z , expressed as Equation 4 (used in stages 2 and 3), is calculated by gearshift force F_S , clutch diameter d_{KS} , teeth angle β and friction μ_{lt} between gearshift sleeve and synchronization ring [7, 9, 10].

$$T_R = jF_S \frac{d_{ms}}{2} \frac{\mu}{\sin\alpha} \quad (3)$$

$$T_Z = \frac{F_S d_{KS}}{2} \left(\frac{\cos\frac{\beta}{2} - \mu_{lt} \sin\frac{\beta}{2}}{\sin\frac{\beta}{2} + \mu_{lt} \cos\frac{\beta}{2}} \right) \quad (4)$$

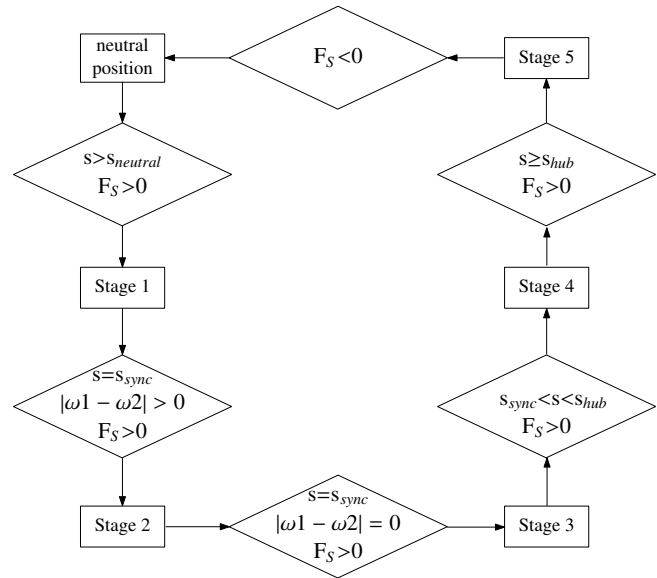


Fig. 6: Flowchart for status determination

2.2.3 Assembly of mechanical submodules

The mechanical subsystem consists of the 3 parts described above (compare Figure 7).

- 1) The gearshifting displacement part, used to simulate the movement of the selector fork
- 2) The synchronization part, functioning to simulate the synchronization process between synchronizer ring and synchronizer hub
- 3) The synchronization status determination and torques calculation part, working to determine the synchronization stages, calculate corresponding friction forces, and coordinate the gearshifting displacement part with the synchronization part

2.3 Modeling Result

Figure 8 shows the relevant physical model. The hydraulic components are modeled with hydraulic library HyLib[®] [11], the mechanical components with Modelica Standard Library (MSL) [12] and some new created blocks based on Modelica[®]. In order to simplify the model structure and improve the model portability, subsystems are built here. For example, *Gear_Selector* is used as a subsystem block, which stands for all mechanical components (see Figure 7).

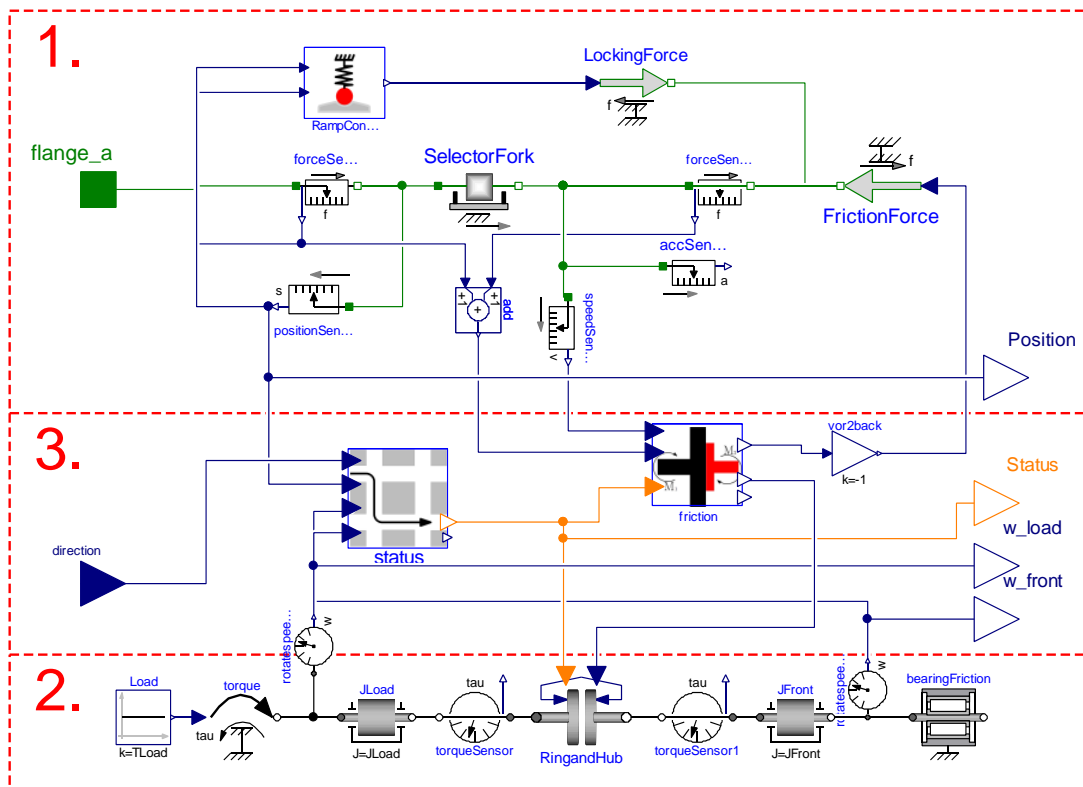


Fig. 7: Mechanical model

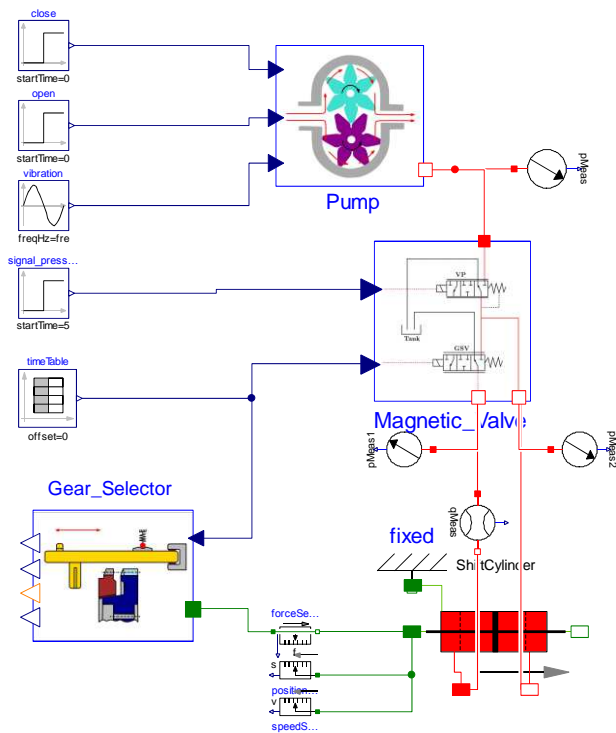


Fig. 8: Synchronization model

3 Testing

In order to verify this dynamic model’s rationality and effectiveness, the following testing steps are carried out:

- 1) testing of the hydraulic model
- 2) testing of the mechanical model
- 3) testing of the whole hydro-mechanical model
- 4) comparison of the simulation results with real AMT test bench measurements

During testing, the dynamic model is driven under an open-loop control. Step- and constant-signals are used for stimulations (see Figure 8).

3.1 Hydraulic Model

The hydraulic supply circuit is first examined against measurement data from real DCT. In this process all magnetic valves are closed, only the oil pump is working. Simulation result, shown in Figure 9 depicts a small model error in comparison to the measurement data, the normalized root mean square error (NRMSE) of $e_{NRMSE} = 4.9\%$. From beginning the pump is kept working until hydraulic pressure reaches the required value. Then the pump stops to wait for restart when pressure level drops, as the result of leakage in the whole hydraulic system, below a predefined threshold value.

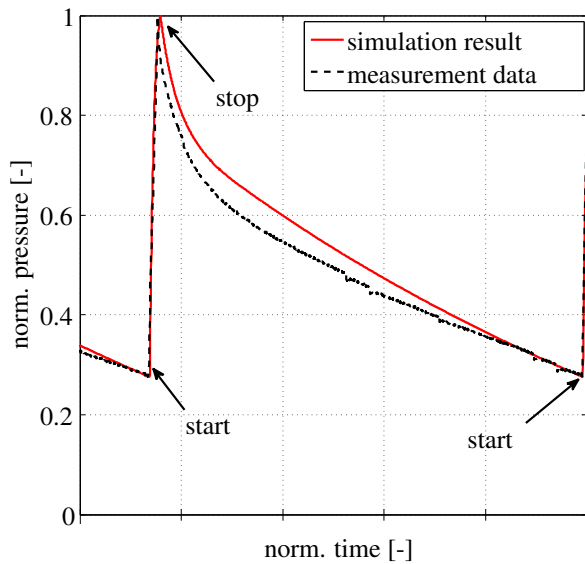


Fig. 9: Comparison of oil pump

Figure 10 shows the movement simulation of the hydraulic gearshift cylinder. In this simulation, the pressure-control valve (*VP1* in Figure 1) is controlled by a constant value while the flow-volume valve (*GSV2* in Figure 1) is controlled by a stimulation signal, as shown in Figure 10 (b). Figure 10 (a) shows change of oil pressures during this process, in which *P1* denotes the oil pressure from the hydraulic pump, *P2* the hydraulic pressure in the right cylinder chamber and *P3* the pressure in the left chamber. *P2* is controlled by *VP1* and the control current is constant; hence *P2* keeps a almost constant pressure value during this process. Figure 10 (b) shows the flow rate into the left cylinder chamber (denoted by *q*, see Figure 1) and the control signal for the flow-volume valve. The constant control signal of *GSV2* (from 0.5 to 1s, from 1.5 to 2.1s) leads to a constant flow rate during the movement of the gearshift cylinder. The displacement process of the cylinder from the middle to right end and reverse is displayed in Figure 10 (c).

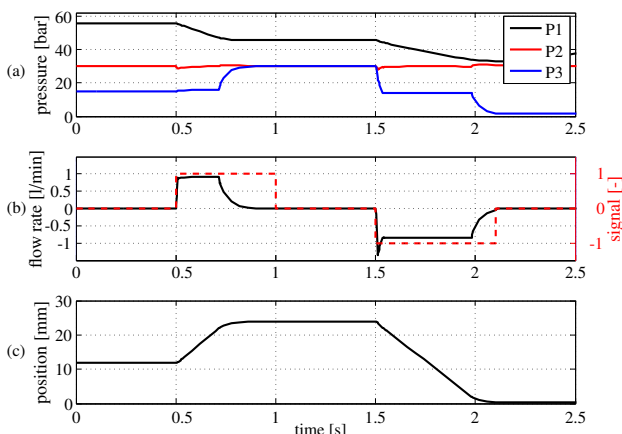


Fig. 10: Simulation results of hydraulic cylinder

3.2 Mechanical Model

This subsection describes the testing of the mechanical model and states that a correct synchronization process can be achieved. Therefore, the typical movement behavior (fast-slow-fast) and the results of the synchronization state determination are examined both.

In Figure 11 the upshifting simulation results are depicted, and its state shows that the model works as expected. Even the *speed difference increases* due to the missing connection between the toothing of the synchronizer hub and ring in stage 4 is also reproduced.

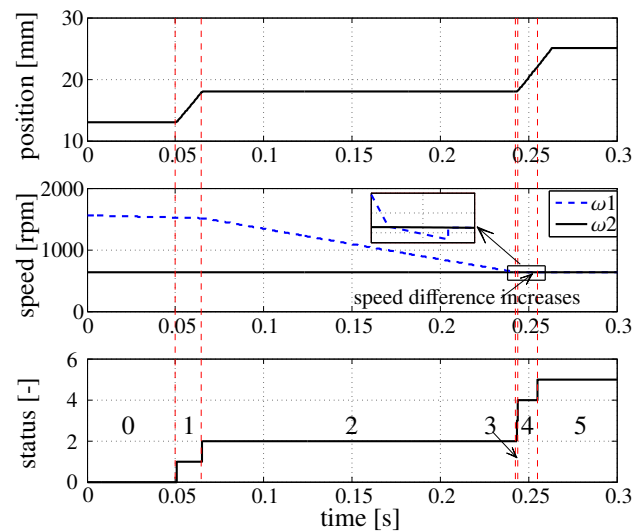


Fig. 11: Simulation results of synchronization

Self-return, an important characteristic of the detent pin (refer to Figure 4), is also tested, see Figure 12. The behavior when shifting force F_C vanishes behind the synchronization point (24mm, upshifting synchronization point is 18mm) is shown on the left-hand side, and the right-hand side shows the behavior of self-return in front of the synchronization point (15mm).

3.3 Hydro-Mechanical Model

Figure 13 shows different synchronizing processes under different working pressures. Synchronization time is reduced as expected when oil pressure increases.

3.4 Comparison with Measurements

Finally, the simulated synchronization process is compared with test bench measurement data from an automated manual transmission (AMT) system (compare [13]) having similar synchronization components. The AMT shifting valves are driven by constant currents

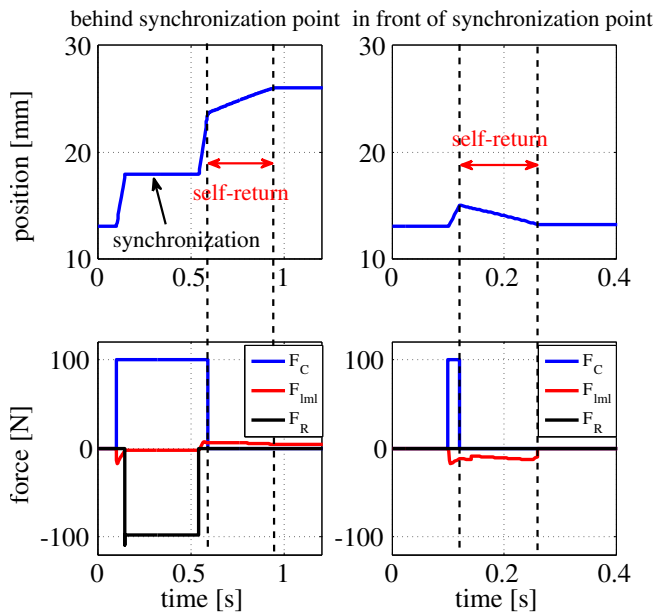


Fig. 12: Simulation results of self-return during up-shifting

and the DCT model shifting valves are driven by step signals. Figure 14 shows the comparison between the representative simulated and the measured shifting processes. The simulation result has a normalized root mean square error (NRMSE) of $e_{NRMSE} = 1.5\%$. It can be stated that the presented model reproduces the characteristic details of the shifting process (pre-sync, locking, unlocking, turning hub and engagement).

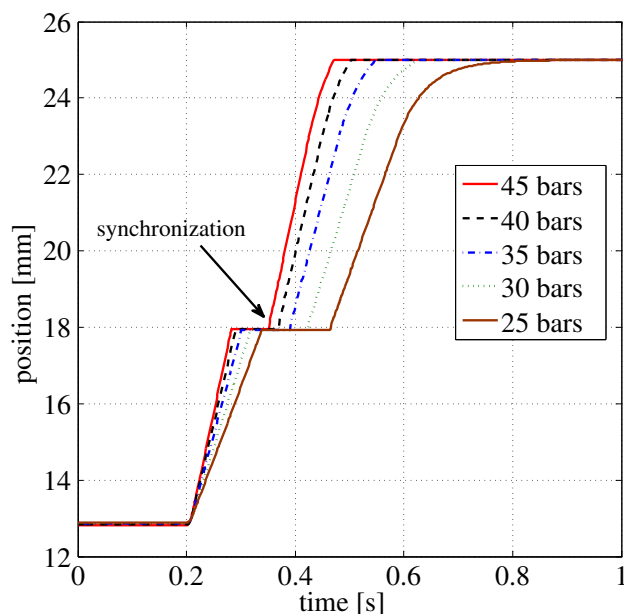


Fig. 13: Synchronization with different pressure

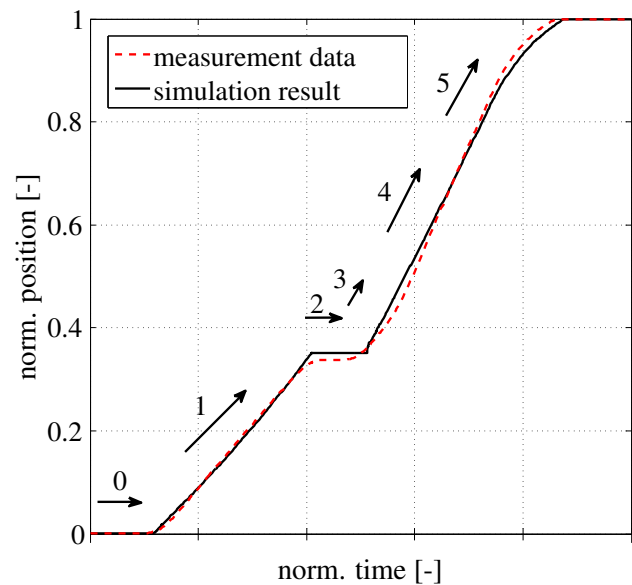


Fig. 14: Synchronization: Comparison of simulation results with measurements

4 Summary and Outlook

This paper gives a detailed introduction to the synchronization process and presents a dynamic Modelica[®] model for the hydro-mechanical actuation and synchronization system based on a popular DCT. This model has following features:

- 1) Gives a detailed representation of the synchronization process with 5 stages instead of simple 3 stages. Additionally in-depth reflection of the nonlinear dynamic system is also presented. This could provide a good reference for shifting quality optimization and more reliable standard for the model-based calibration.
- 2) Reveals the phenomenon that speed difference increases after the synchronization process because of power interruption in this stage. This is important to judge shift quality control strategies because during this phase serious problems as tooth breaking and shifting noise may occur.
- 3) Presents the user a fundamental understanding of the components composition principle and the system working function.
- 4) Shows that the tested hydraulic and mechanical modules have a good modularity for other similar system setups only through parameters changes.
- 5) Provides a good platform for the model-based calibration and function development.

Based on this dynamic simulation model, follow-up researches become possible: such as the integration of

a clutch system (refer to [14]) and an appropriate control algorithms into a complete transmission model. The further important research field of model-based calibration on AMTs and DCTs in order to optimize shifting quality can also be identified.

References

- [1] B. Wede. *Modellierung eines 6-Gang-Schaltgetriebes mit Hilfe der Modellierungssprache Modelica*. GRIN Verlag GmbH, 2010.
- [2] U. Schreiber, J. Schindler, and E. Steinmetz. *Systemanalyse in der KFZ-Antriebstechnik, Objektorientierte Modellbildung und Simulation kompakter KFZ-Antriebsstränge*. 6. Band. Expert Verlag, Renningen, 2001.
- [3] C. Gühmann. Model-based testing of automotive electronic control units. In *3rd International Conference on Materials Testing*, Nürnberg, 2005.
- [4] M. Schäfer, A. Damm, Th. Pape, and etc. The control unit of volkswagen's new dual-clutch transmission. In *6. Internationales CTI-Symposium Innovative Fahrzeug-Getriebe*, Berlin, 2007.
- [5] Volkswagen. *Self-study Programme 390: The 7-speed Double-clutch Gearbox 0AM*. Volkswagen AG, 2007.
- [6] G. Lechner and B. Bertsche. *Automotive Transmissions: Fundamentals, Selection, Design and Application*. Springer, 1999.
- [7] S. Nowoisky, R. Knoblich, and C. Gühmann. Comparison of different model types based on a synchronization of an automated manual transmission. In Clemens Gühmann, Jens Riese, and Thieß-Magnus Wolter, editors, *Simulation und Test für die Automobilelektronik IV*, page 38...47, Wankelstra13 D-71272 Renningen, 2012. Expert Verlag.
- [8] INA. *Detent Pins for Automotive Transmissions*. Schaeffler Technologies AG & Co. KG, 8 2007.
- [9] INA. *Intermediate Rings for Multi-Cone Synchronizer Systems*. Schaeffler Technologies AG & Co. KG, 8 2007.
- [10] E. Kirchner. *Leistungsübertragung in Fahrzeuggetrieben*. Springer, 2007.
- [11] Hylib (2009), version 2.7. Product help, Modelon, 2009.
- [12] Modelica standard library (2008), version 3.1. Product help, Modelica Association, 2008.
- [13] R. Knoblich, J. Beilharz, and C. Gühmann. Modellbasierte steuergeräteentwicklung für kfz-getriebesysteme am prüfstand. In *Tagungsband Mechatronik 2011, Dresden, 31.Maerz - 1.April 2011*. T. Betram, B. Corves, K. Janschek, 2011.
- [14] S. Nowoisky, C. Shen, and C. Gühmann. Detailed model of a hydromechanical double clutch actuator with a suitable control algorithm. In *Proceedings of the 8th International Modelica Conference*. The Modelica Association and Fraunhofer Institute for Integrated Circuits IIS; Design Automation Division EAS, 2011.

Predicting the launch feel of automatic and dual clutch transmissions

Neil Roberts Mike Dempsey

Claytex Services Ltd.

Edmund House, Rugby Road, Leamington Spa, CV32 6EL

neil.roberts@claytex.com

mike.dempsey@claytex.com

Abstract

The Powertrain Dynamics Library (PTDynamics) has been developed using a new approach to modelling the mechanics of rotating MultiBody systems. This paper will highlight the recent developments within the PTDynamics library with a focus on the dynamic torque converter and wet clutch models that enable the prediction of the launch feel of automatic and dual clutch transmission equipped vehicles. Two examples are presented: one that compares the effect of oil temperature on the initial launch of a vehicle with a dual wet-clutch transmission; and a second that compares the behaviour of steady state and dynamic torque converter models.

Keywords: powertrain dynamics, driveability, dynamic torque converter, wet clutch, automatic transmission, dual-clutch transmission

1 Introduction

The transmission and driveline of a vehicle have a large influence on the customer driving experience and perception of quality, as well as the efficiency and performance of the vehicle. The influence of hybridization within a vehicle has greatly increased the architecture variants available to vehicle manufacturers and consequently has complicated the selection of the most efficient hardware solution.

The Powertrain Dynamics (PTDynamics) library has been developed as a commercial Modelica library to aid evaluation of the many technology and topology options. It also provides the capability to model powertrain systems in sufficient detail to support the design and validation of the associated control systems and to optimize the vehicle's response to driver inputs.

The initial application of the library has been the transmissions and drivelines within automotive applications but it can be applied to any powertrain system. This paper explores some of the recent additions to the library that are used in the simulation of vehicle transients such as initial launch, tip-in and

tip-out and gear shifting. Two examples are presented illustrating how the new additions enhance the level of detail that can be included in models to predict the initial launch of vehicles with automatic and dual clutch transmissions.

2 Powertrain Dynamics Library

2.1 Overview

Transmission and driveline systems comprise a number of key components that influence their dynamic behaviour and efficiency. The PTDynamics library has been developed to provide models for all of these components and assemblies as easy to use MultiBody models. The design objective is to make it easy to assemble a MultiBody powertrain model and achieve good simulation performance and results without having to develop a detailed knowledge of Modelica.

The range of components included in the first version of the PTDynamics library and the fundamental approach used to model the mechanics are described in [1]. This range of components is continually enhanced and refined with this paper describing some of the more significant recent developments.

2.2 Dynamic torque converter

In automatic transmissions the engine and gearbox are coupled by a torque converter. This is typically modelled using the steady state performance curves for the torque converter that relate speed ratio, torque ratio and capacity factor (k-factor, MPC2000, or c-factor), see Figure 1 for an example of these curves. These curves are readily available from the torque converter manufacturers and make it relatively easy to implement a steady state torque converter model. Most simulation tools only offer this type of steady state torque converter model that works well for drive cycle studies but is inadequate for the simula-

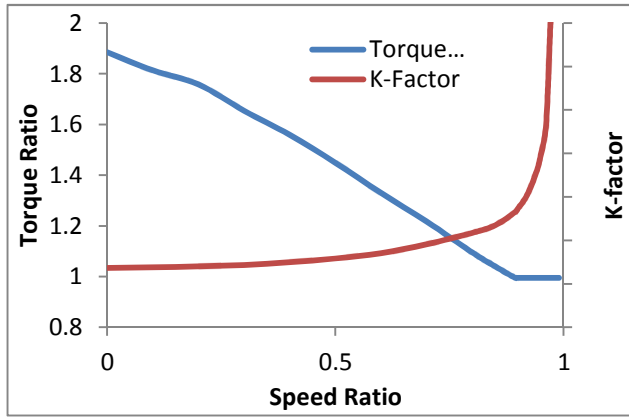


Figure 1: Steady state torque converter performance curves (speed ratio, torque ratio and k-factor)

tion of transient events such as launch, tip-in, tip-out or gear shifting.

The problem is that models based on these curves cannot capture the transient behaviour of the torque converter which has a significant impact on the driving experience. During large transient events such as initial launch, gear shifting and driver tip-in and tip-out events the transient response of the torque converter has an impact on the vehicle response and the perception of performance experienced by the driver.

A dynamic torque converter model has been implemented to overcome this problem and enable the torque converters fluid inertia and stator dynamic behaviour to be included in simulations. The model is based on the nonlinear lumped parameter model derived in Hrovat and Tobler [3] that describes the converter dynamics. It has been implemented to fit within the PTDynamics framework for a torque converter model which means that the user can very easily switch between an existing steady state torque converter model and the new dynamic torque converter model.

The basic layout of a 3 element torque converter is shown in Figure 2 with the key parts identified. The impeller is connected to the engine, the turbine is connected to the gearbox and the stator is connect-

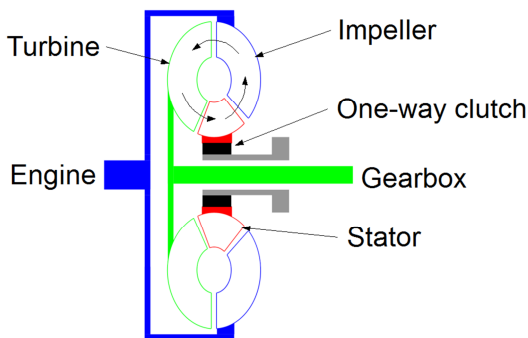


Figure 2: Schematic of a 3 element torque converter showing the fluid flow direction in the driven condition

ed to the gearbox housing via a one-way clutch. Energy is transferred between these 3 components by the hydraulic fluid within these control volumes (impeller, stator, turbine).

The moment-of-momentum equation is applied to each of these control volumes and relates the rotational velocity of the mechanical components and the torque to a fluid flow velocity along the torque converter rotational axis. This results in a single first order state equation for each element and for the impeller this gives the following equation:

$$I_i \dot{\omega}_i + \rho S_t \dot{Q} = - \left(\omega_i R_i^2 + R_i \frac{Q}{A} \tan \alpha_i - \omega_s R_s^2 - R_s \frac{Q}{A} \tan \alpha_s \right) Q + \tau_i \quad (1)$$

This equation relates the speed of the impeller (ω_i), torque on the impeller (τ_i), its radii at the centre of its outlet port (R_i), the angle of the blade surface to the normal (α_i) and the fluid volume flow rate (Q) is related to the conditions at its input from the stator. The state equations for the turbine and stator are of a similar form.

The fluid state equation links the relationship between the fluid volume flow rate (Q) and the mechanical inertia velocities ($\omega_{i,t,s}$) using a conservation of momentum energy balance given by:

$$\rho (S_i \dot{\omega}_i + S_t \dot{\omega}_t + S_s \dot{\omega}_s) + \frac{\rho L_f}{A} \dot{Q} = \rho (R_i^2 \omega_i^2 + R_t^2 \omega_t^2 + R_s^2 \omega_s^2 - R_s^2 \omega_i \omega_s - R_i^2 \omega_s \omega_t - R_t^2 \omega_s \omega_t) + \omega_i \frac{Q}{A} \rho (R_i \tan \alpha_i - R_s \tan \alpha_s) + \omega_t \frac{Q}{A} \rho (R_t \tan \alpha_t - R_i \tan \alpha_i) + \omega_s \frac{Q}{A} \rho (R_s \tan \alpha_s - R_t \tan \alpha_t) - p_L \quad (2)$$

Where the p_L term represents the losses in the familiar form of shock losses from non-ideal flow conditions and fluid friction losses. These are defined as shock velocity coefficients ($C_{sh,i,t,s}$) and a fluid friction factor (f).

$$p_L = \frac{\rho}{2} \text{sgn}(Q) (C_{sh,i} V_{sh,i}^2 + C_{sh,t} V_{sh,t}^2 + C_{sh,s} V_{sh,s}^2) + \frac{\rho f}{2} \text{sgn}(Q) (V_i^{*2} + V_t^{*2} + V_s^{*2}) \quad (3)$$

These equations fully characterize the dynamic behaviour up to sufficiently large frequencies (~50Hz) to model fast transient phenomena occurring during throttle steps and rapid speed ratio changes.

Due to the ‘free body’ formulation approach taken, the model relies upon knowing some key internal geometry parameters of the torque converter; most notably the radii and blade angles that are not normally quoted/released by torque converter manufac-

tures. These parameters have to be calibrated before the dynamic model can be used and this is done in two stages using the Optimisation toolbox available for Dymola.

The first stage of the optimization process is to tune the model parameters so that the dynamic torque converter model accurately predicts the steady state performance. This is achieved by running the torque converter under steady state conditions and comparing the quoted steady state performance curves with the simulation results. After the optimisation of the parameters to match the steady state response, additional experimental data captured under transient driving conditions is required to calibrate the dynamic response of the torque converter model. This approach does allow the user to tune these design parameters to obtain good agreement with experimental data.

2.3 Wet clutches

Wet clutches are key components in both automatic and dual-clutch transmissions and a new model for predicting the torque response of a wet clutch pack has been developed. The torque across a wet clutch is a direct function of automatic transmission fluid (ATF) film thickness, pressure distribution and asperity pressure at the interface. The model calculates the total torque across the wet clutch as the sum of the hydrodynamic torque and asperity torque.

The hydrodynamic torque is created early in the clutch engagement phase through fluid shear with the hydraulic pressure supporting the normal load and preventing physical contact of the clutch plates. As the film thickness decreases to a similar magnitude to the surface roughness of the friction plates, the asperities of the friction material make contact, supporting the normal load on the clutch and reducing the fluid hydrodynamic torque to zero. The asperity torque then determines the total torque transfer. It is these phenomena that heavily influence the torque characteristics during a clutch engagement.

The hydrodynamic torque is based on the ATF film thickness (h) calculated using an approximate Reynolds equation for a rough and permeable surface which has been shown to be very similar to the full modified Reynolds equation [4]. The contribution of the hydrodynamic pressure (ξ), material permeability (δ), surface roughness (g) and the real contact area (A_{red}) is given by:

$$\frac{d\hat{h}}{dt} = \frac{\phi(\hat{h})\xi(\hat{h})\delta(\hat{h})}{g(\hat{h})A_{red}} \gamma \hat{h}^3 \quad (4)$$

The normalized oil film thickness $\hat{h} = h/h_0$ is used directly in the hydrodynamic torque calculation:

$$T_h = \mu N_f (\phi_f + \phi_{fs}) \int_0^{2\pi} \int_{r_i}^{r_o} \frac{r^2 \omega_{rel}}{h} r dr d\theta \quad (5)$$

where h_0 is the steady state oil film thickness, and the pressure and shear stress flow factors (ϕ_f , ϕ_{fs}) from Patir and Cheng [5] account for flow between rough surfaces. The kinematic viscosity of the fluid (μ) is calculated using the ASTM D341 standard [6] as:

$$\log(\mu + 0.7) = A - B \log(T) \quad (6)$$

where A and B are two coefficients calculated from two known viscosity-temperature operating points of the ATF.

The asperity torque is calculated from the friction coefficient (μ_f), number of friction surfaces (N_f), clutch radii (r_i, r_o), and the applied pressure (P_a).

$$T_a = \mu_f N_f \int_0^{2\pi} \int_{r_i}^{r_o} r^2 P_a dr d\theta \quad (7)$$

The applied pressure on the asperities on a rough surface is considered to be proportional to the area in contact and the Young's modulus of the friction material.

Due to the dependency on the ATF film thickness on clutch torque, under multiple engagements the time taken for the oil film to be replenished after an engagement would affect the torque profile for the next engagement.

As no description for this film replenishment phase seems to be available in published literature, an exponential rise time has been introduced to include the effects of multiple engagements on the oil film thickness with a parameter $riseTime$ to describe the time taken for the film thickness to return to its pre-engagement full film thickness (h_0):

$$\left(\frac{1-h_0}{riseTime} \right) \left(\exp \left(-\frac{time-Htime}{riseTime} \right) \right) \quad (8)$$

This models ability to account for the hydrodynamic torque contribution enables the significant thermal effects to be accounted for within the torque response in wet clutches; a common cause for negative feedback on dual clutch transmissions particularly in low temperatures at initial launch due to the high oil viscosity. This also provides a more detailed description of the real system to enable calibration of control strategies during clutch slip control and engagement.

The availability of parameter data for the clutches of interest such as the lining thickness and permeability as well as the availability of a thermal model

that can account for the thermal performance of the system are the two significant limiting factors for the prediction of wet clutches.

2.4 Aggregated shafts

In the PTDynamics library, an aggregated shaft method has been developed to model the cardan shafts and joints within a driveline. The kinematic relationship of the shaft and its associated joints is described using a single aggregated joint between the two ends. Figure 3 shows an example of a shaft with a joint such as a constant velocity joint at each end of the shaft. Using this approach the shaft itself can be considered to have a fixed or variable length.

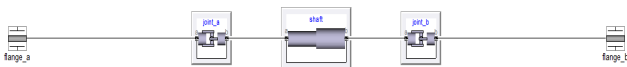


Figure 3: Diagram of an aggregated shaft with a joint at each end

This approach is consistent with the aggregated joint approach in the Modelica MultiBody library [7] where the removal of the constraint equations eliminates the nonlinear equations generated and the motion equations are solved analytically to enhance the simulation performance. Figure 4 shows how this is implemented in the PTDynamics library with the degrees of freedom for both joints being modelled in the special joint shown at the bottom of the diagram.

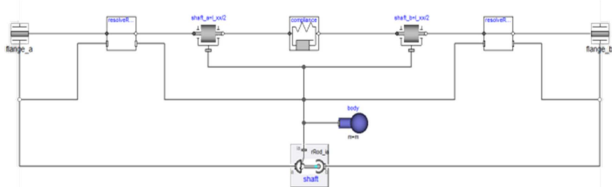


Figure 4: Internal diagram of an aggregated shaft model

One problem that can be introduced by these aggregated joints is that the MultiBody frames in the related connectors can appear to be rotated at 180 degrees relative to one another. This would make any resulting rotation that is tracked in the flange connector appear to be in the wrong direction at this point in the model (see [1] for further details on the basic Rotation3D methodology). To handle this we use special blocks that resolve the rotation direction in the flange connector to make sure that it is consistent with the orientation of the bearingFrame wherever such a rotation is possible in a component model. The resolve rotation blocks are used at both ends of the aggregated shaft model.

The topology of the shaft model can remain constant for both plunging and fixed length shafts with a

simple replacement of the shaft joint at the bottom of the diagram shown in Figure 4.

The torsional compliance of the shafts in the driveline play a key role in the longitudinal response of the vehicle, with the driveshafts and propshafts often containing the largest proportion of the total compliance within the system. A compliance model is therefore included within the central shaft component and within both joints as shown in Figure 3. This compliance model can be configured to be Rigid, Linear, Linear with Backlash, Nonlinear and Nonlinear with plastic deformation to cover the different use cases and model fidelities required for driveline testing.

To aid bringing simulation earlier into the design cycle and extend the usability of the library, a number of shaft options have the ability to estimate the mechanical properties (stiffness, mass and inertia) by entering simple geometry and material properties. This can ease the burden of knowing many parameters not available early in the design stage and where simple torsion theory using geometry can yield reasonably accurate results.

3 Vehicle Systems

The components described have been used to model two different powertrain configurations. Built using the templates provided in the PTDynamics library, they maintain the same high level vehicle architecture but they represent very different physical systems. The template approach is based on the VehicleInterfaces library [2]. Within the PTDynamics library this architecture structure has been extended to provide templates for common transmission and driveline arrangements.

Two different powertrain architectures are considered: first, a mid-engined rear wheel drive trans-axle vehicle with a dual clutch transmission; second, a front-engined rear wheel drive vehicle with an automatic transmission and torque converter.

These examples both represent a car with a mass of 1500kg using a chassis model with pitch, bounce and roll degrees of freedom as well as the longitudinal motion. A mean-value engine model developed using the Engines Library [1] is used in both cases: for the four-wheel drive vehicle we use a V6 engine and for the rear-engined vehicle an inline 4 cylinder engine is used. The engine and transmission assemblies are mounted in the vehicle body using an elastomeric mount with a linear force-displacement characteristic.

3.1 The Transmission

The transmissions are built using templates from the PTDynamics library as shown in Figure 5. These templates split the gearbox into 3 main sub-systems, the engagement device, the gearset and the gear selection mechanism. An engagement device in the form of a clutch assembly or torque converter sits between the engine and the gearset. The gearset includes the gears, shafts, bearings and synchronisers or clutches used to engage different gears. The gear selection mechanism defines the actuation system that translates the driver movement of the hand lever or control system gear demand into actuation of a clutch or synchroniser. This system architecture suits many applications commonly seen in automotive transmissions.

The dual clutch transmission comprises two concentric wet clutches with a three shaft type gearset as shown in Figure 6. Simple synchroniser models for each gear are included to enable the gearbox to be used to run tests in different gears but the detailed shift dynamics are not currently included and will be introduced in a future development of the PTDynamics library.

The automatic transmission is a 6 speed gearbox consisting of a front Epicyclic and a rear Ravigneaux gearset with 2 brakes and 3 clutches to control the overall gear ratio (see Figure 7). The gearset is coupled to the engine via a torque converter. The speed and torque dependent losses are lumped for convenience and based on the current gear signal. Where the data is available the losses can be distributed to the appropriate bearings and gear mesh models.

3.2 The Driveline

A range of templates for commonly occurring driveline configurations are provided in the PTDynamics library. The example in Figure 8 illustrates one of the four wheel drive templates that is available. In this case the driveline includes a central differential that is mounted to the transmission case. The front and rear differentials are independently, elastically mounted within the vehicle body. All of the components are replaceable so that the user can select the appropriate model for their application.

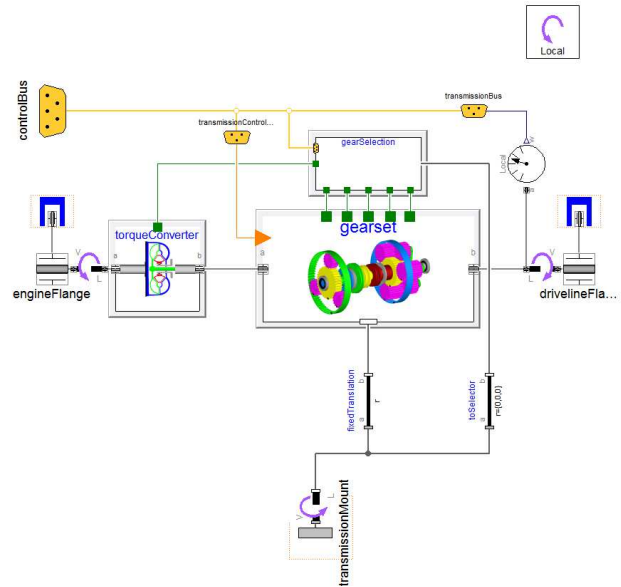


Figure 5: Automatic transmission used in the vehicle model

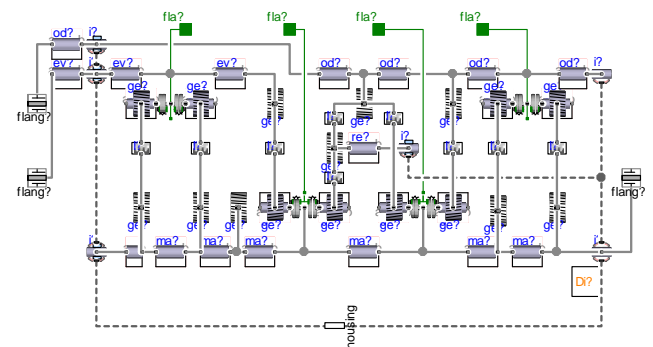


Figure 7: Dual clutch transmission 3 shaft gearset

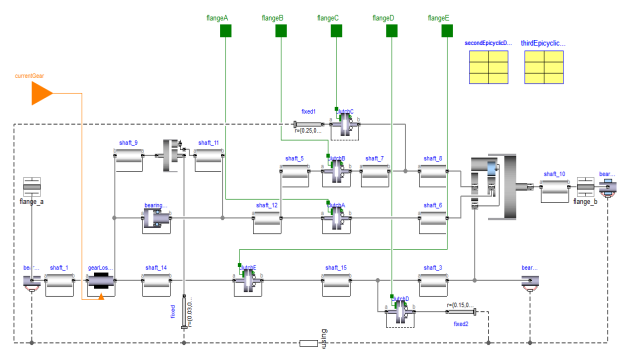


Figure 6: Gearset model for the automatic transmission model

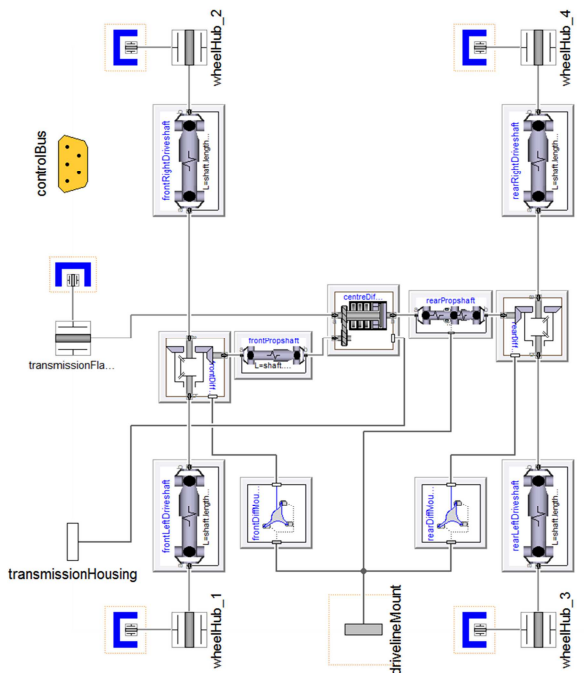


Figure 8: Four wheel drive driveline template with centre differential

4 Results

4.1 Test definition

The two powertrain examples were used to model a vehicle launch from standstill in 1st gear. In these tests the engine starts at idle speed and we are interested in the vehicle longitudinal response which is what the driver will experience. We will focus on the behaviour and influence of the engagement devices (i.e. torque converter and wet clutch) on the longitudinal acceleration.

4.2 Vehicle with dual clutch transmission

The vehicle model used for these experiments is a mid-engined car with a 7 speed dual-clutch transmission and integrated rear differential. The model includes all effects that influence the initial launch of the vehicle such as the power-unit mounting system, tyre slip, suspension (including the fore-aft compliance) and the torsional compliance of all the shafts.

This example will look at the effect of oil temperature on the initial pull-away of the vehicle. With the oil in the wet clutch at the normal operating temperature the pull-away of this type of vehicle will be calibrated to deliver the acceleration profile that best matches the brand image of the manufacturer. This could result in a very smooth pull-away or be calibrated to give a more aggressive start with a higher

jerk at the start of the launch. However the start is calibrated, the aim for the manufacturer is for this to be consistent at all operating temperatures of the clutch. At low temperatures though, this becomes more difficult to achieve due to the change in viscosity of the oil. This change is usually obvious to the driver because the launch will not be as smooth and a lot of effort with the calibration is required to minimise the effect.

The launch test is a gentle acceleration with the driver requesting a small amount of the available engine torque resulting in the vehicle accelerating to just 20kmh in 5 seconds. Figure 9 shows the results of this pull away for the cold and warm gearbox tests. In the case of the warm pull away the accelera-

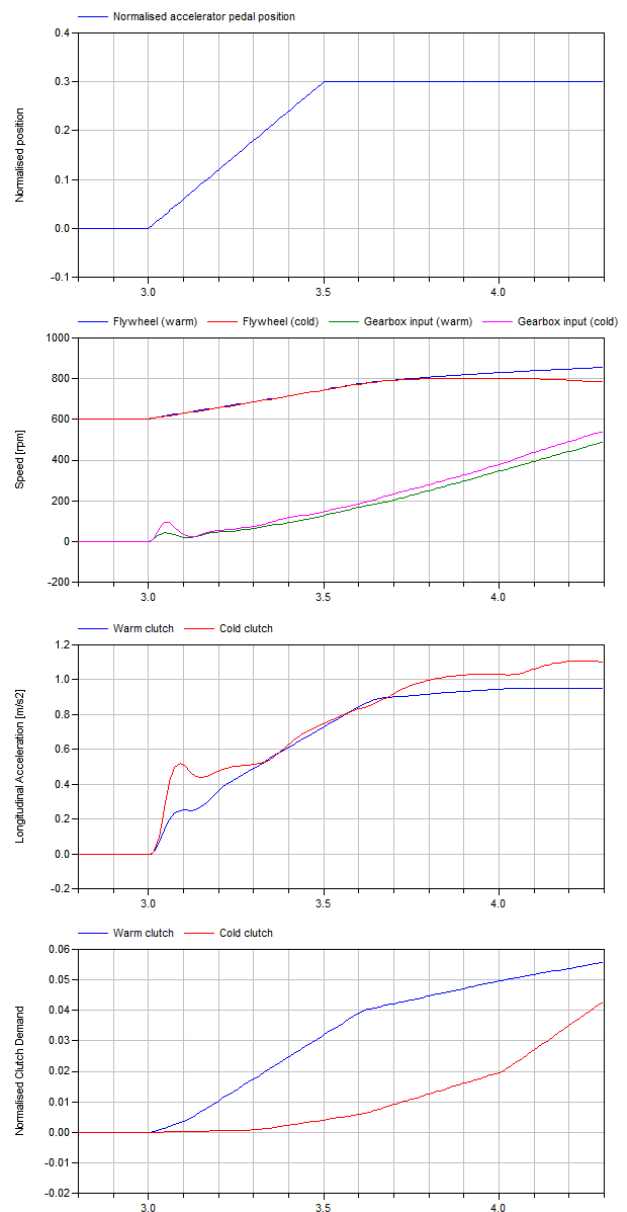


Figure 9: Comparison of pullaway with warm and cold oil in the wet clutch. Top is the driver accelerator demand; 2nd plot is the engine and gearbox input speeds; 3rd is the longitudinal acceleration; and bottom is the clutch demand

tion builds very smoothly.

With the cold gearbox though a smooth pull-away is not achieved even though the clutch demand is significantly reduced due to the low oil temperature. The bottom plot in Figure 9 shows the change in the clutch demand between the warm and cold oil temperatures. Despite the large reduction in the clutch demand during the first 0.2s, when the oil temperature is low we still get a relatively large acceleration as soon as the clutch pressure begins to rise. This is due to a large amount of torque that gets generated as soon as the fluid layer begins to be compressed resulting in a torque spike and corresponding longitudinal acceleration.

How the driver judges the driveability performance of a vehicle is often related to 3 objective variables: delay time; peak acceleration and jerk [8]. Looking at these 3 objective measures using Figures 9 and 10 we can interpret the vehicle response and compare the cold and warm performance. There is no change in delay time but there is a big increase in the jerk (See Figure 10) and a related change in the acceleration profile (see Figure 9). For the cold pull-away event, the jerk is 2.5x higher than with a warm clutch. This will all effect the drivers perception of how smooth the car is.

To cope with the low oil temperature the clutch engagement profile has to be reshaped as well as reducing the actual applied pressure during the early phases of the engagement.

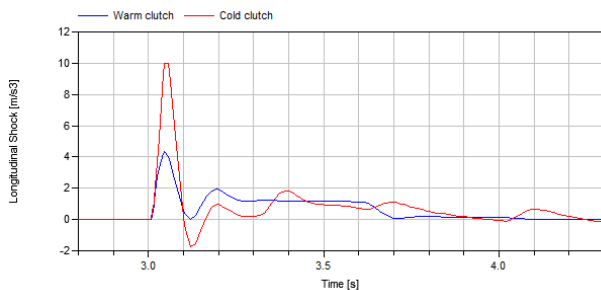


Figure 10: Longitudinal jerk during pullaway

4.3 Vehicle with automatic transmission

The vehicle model used for these experiments is a front-engined, rear-wheel drive vehicle. It is fitted with a 6 speed automatic transmission with a torque converter and lock-up clutch. The model includes all effects that influence the initial launch of the vehicle such as engine and differential mounting systems, tyre slip, suspension (including the fore-aft compliance) and the torsional compliance of all the shafts.

The torque converter model can be easily changed between a steady state model and the calibrated dynamic model. This analysis focuses on the detailed differences in the vehicle response due to

the use of a steady state and dynamic torque converter model.

The experiment is a launch from rest with the driver releasing the brake pedal and then applying the accelerator pedal. The rate of pedal actuation is the same in both tests and the engine is running at idle speed with first gear engaged in the transmission at the start of the test. Therefore the only difference between the two tests is the torque converter model.

Figure 11 shows the normalised driver pedal posi-

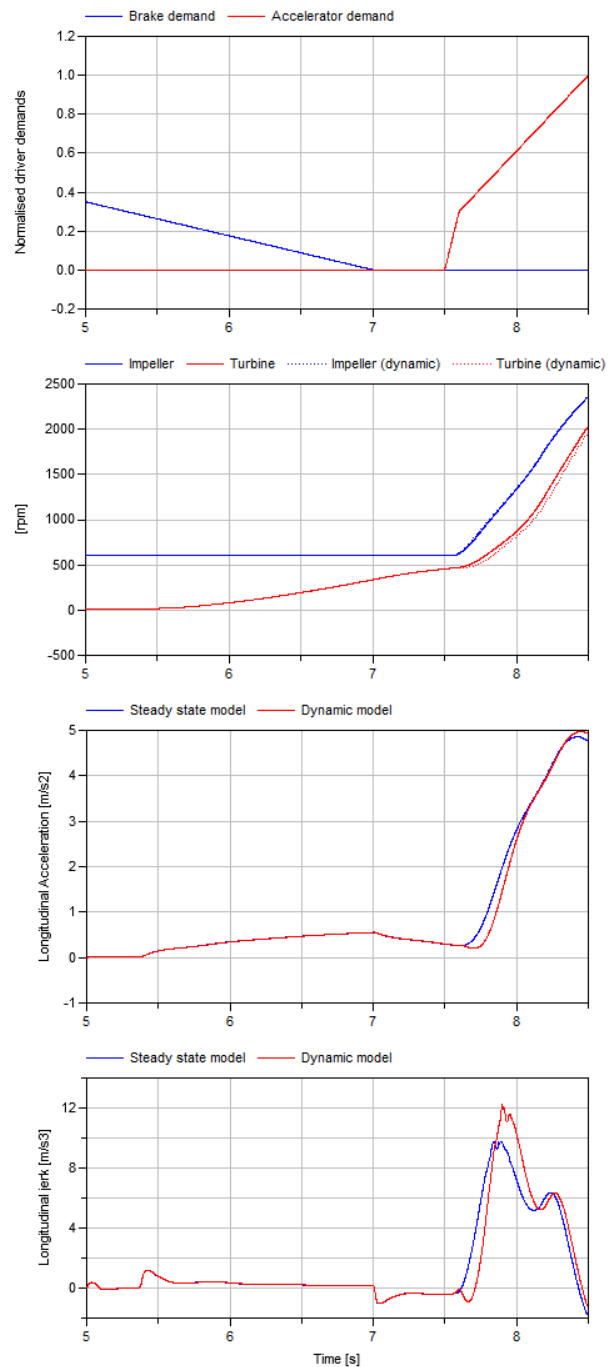


Figure 11: Pullaway comparing the steady state and dynamic torque converter models. Top: driver demands; 2nd plot is the impeller and turbine speeds with the dynamic model results in dashed lines; 3rd is the longitudinal acceleration; Bottom is the longitudinal jerk

tions (top graph) aligned with the impeller and turbine speeds (middle graph) and the vehicle longitudinal acceleration (bottom graph).

The longitudinal acceleration of this vehicle can be broken down into two phases. Phase 1 occurs between 5.0 and 7.0s while the brakes are slowly released and phase 2 begins as the driver steps across from the brake pedal to the accelerator pedal.

During phase 1 the acceleration profile is dominated by the release characteristics of the brake system. This is because while the vehicle is held stationary the torque converter is applying torque to the gearbox input. As soon as the friction torque in the brakes reduces below a certain level the vehicle will start to creep forward. This model includes a very simple brake system and so the brake pedal release profile is modified to limit the acceleration during phase 1.

Phase 2 of the launch is where we see the differences between the steady state and dynamic torque converter models. At this stage in the test the driver is quickly applying the accelerator pedal to demand 100% torque from the engine and it is during the time 7.5 to 8.0s that we see the effect of the torque converter model on the results.

With the dynamic torque converter model we see an increased delay between the driver demand and the vehicle acceleration combined with an increase in the jerk once the dynamic model starts to accelerate. Both of these metrics are known to influence the drivers perception of driveability [8].

Accurate prediction of these driveability metrics together with other measurements such as fuel usage, emissions and thermal effects enable the launch strategy within the engine control software to be adjusted and calibrated to deliver the desired balance between vehicle performance feel, fuel economy and emissions.

5 Conclusions

An overview of the developments made within the Powertrain Dynamics Library is presented and focused on the prediction of initial launch for two types of vehicle. In the first case, the effect of oil temperature on the initial launch of a dual wet-clutch transmission equipped vehicle is presented followed by a look at how a dynamic torque converter model can improve the accuracy of the initial launch prediction of an automatic transmission equipped vehicle.

Two key areas for transmission modelling have been addressed through the introduction of more detailed wet clutch models and a dynamic torque converter model. These enable more dynamic driving

events such as launch and gear shifting to be modelled and accurately predicted using Modelica based models.

References

- [1] Dempsey M., and Picarelli A. Investigating the MultiBody Dynamics of the Complete Powertrain System. Como, Italy: Proceedings 7th Modelica Conference, 2009.
- [2] Dempsey M., Gäfvert M., Harman P., Kral C., Otter M., and Treffinger P., Coordinated automotive libraries for vehicle system modelling, Vienna, Austria, Proceedings of the 5th International Modelica Conference, 2006.
- [3] Hrovat D., and Tobler W.E., Bond-graph modelling and computer simulation of automotive torque converters. Journal of the Franklin Institute 319,93-114, 1985
- [4] Yang Y., Lam R., and Fujii T., Prediction of Torque Response During the Engagement of Wet Friction Clutch. SAE Technical Paper, 981097, 1998.
- [5] Patir N., and Cheng H., Application of Average Flow Model to Lubrication Between Rough Sliding Surfaces. ASME Journal of Lubrication Technology, 101, 220, 1979
- [6] American Society for Testing and Materials (ASTM) International. D341 -09 Standard Practise for Viscosity-Temperature Charts for Liquid Petroleum Products, Accessed 08/05/2012
<http://www.astm.org/Standards/D341.htm>
- [7] Otter M., Elmqvist H., and Mattsson S.E.: The New Modelica MultiBody Library, Linköping, Sweden, Proceedings of the Modelica 2003 Conference, 2003
- [8] E. Cacciatori, Advanced Control Concepts for a Parallel Hybrid Powertrain with Innitely Variable Transmission," Ph.D. dissertation, Cranfield University, Cranfield, 2007.

Modelling of Elastic Gearboxes Using a Generalized Gear Contact Model

F.L.J. van der Linden, German Aerospace Center (DLR)
Münchner Straße 20, 82234 Weßling, Germany
franciscus.linden@dlr.de

Abstract

The object of this paper is to present an universal model that describes the gear contact between two gears in a planar environment. The model includes elastic effects between the gear wheels. Using this model it is possible to create arbitrary spur gear connections as well as all kinds of epicyclic gearing configurations by supplying the proper external constraints. The presented model is implemented in the Modelica language and Dymola is used for the simulations.

Keywords: Elasticity, Gearbox, Epicyclic Gearing, System Modeling

1 Introduction

Gear transmissions are widely used in almost all engineering applications. These range from cheap plastic consumer printers, aircraft actuators up to high precision positioning drive systems. The design of these transmissions is dependent on the application. This design process ranges from "looking up a standard gear in a catalog and hope it will work" up to detailed dynamic analysis using Finite Elements Methods.

At the moment gear research is mainly focused on the understanding of gearboxes. Özgüven and Houser [4] wrote a model review in 1988, Parey and Tandon [6] did the same in 2003. These works present a good overview of the work done up till that time. More recent works can be sorted into 3 groups:

1. Rigid models or simple elastic systems with only rotational degrees of freedom [7, 3]
2. Coupled torsional and transversional elastic models [9, 1, 8, 5]
3. Self excited gears models; gear eccentricity, transmission errors and stiffness variations [3, 1, 9, 5]

Some of these mentioned works have friction effects included. Most of the recent works include a full transversional-torsional coupled model including either detailed friction effects or self excitation. There is a clear trend on an increasing model detail and complexity.

However, all the models above, are not flexible when gearing configurations like compound planetary gears or even more exotic configurations are used. In the pre-design stage of such a gearbox, reduction ratios as well as internal vibrations are usually important. In this paper a model will be presented that can simulate arbitrary elastic gearbox configurations by relying on a planar library. This approach makes it very easy to evaluate several model configurations without a lot of design work. To keep the simulation time low, the presented model does not include any friction effects, since they are often not directly necessary in the pre-design stage.

2 Gear Forces and Equations

In this chapter the forces and torques on the gear wheels are evaluated. Since these forces and torques differ for internal- and external toothing, these aspects are treated as separate cases.

2.1 Force and Moment balance of external toothing

In Figure 1 a schematic overview of two gear wheels in contact are shown. The rotation of the gear wheels are ϕ_A and ϕ_B , shown by the angles to the body-fixed red and blue markers on the gear wheels.

The gear ratio is defined by:

$$\frac{r_A}{r_B} = -i \quad (1)$$

This ratio is constant for each gear angle and position.

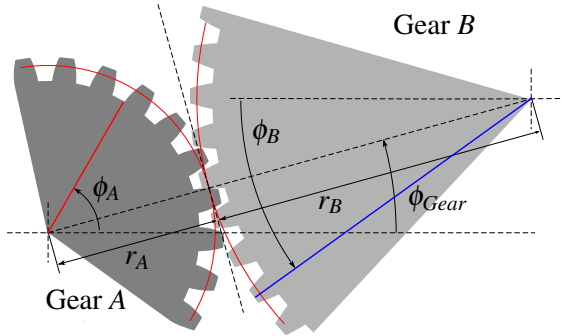


Figure 1: Schematic overview of two gearwheels in contact. The blue and red line are fixed markers on the gear wheels. In the figure $\dot{\phi}_A > 0$ and Gear A drives Gear B.

Figure 2 shows a free body diagram of the two gears in contact. The forces of only one contact point are displayed.

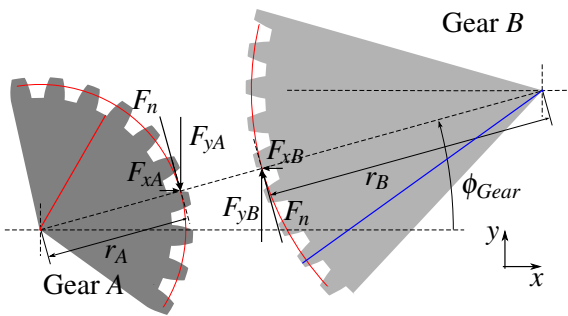


Figure 2: Free body diagram of the two gearwheels from Figure 1.

Using Figure 2, it is possible to create the torque and force balances of each gear wheel for external toothing configurations. These forces and torques are resolved in the fixed coordinate system shown in Figure 2. The use of a fixed coordinate system and gear angle ϕ_{gear} makes it possible to use the contact model also in more complex gear systems (e.g. all kinds of Epicyclic gearing configurations).

$$\tau_A = F_n r_A$$

$$\tau_B = F_n r_B$$

$$F_{xA} = -\sin(\phi_{gear}) F_n$$

$$F_{yA} = \cos(\phi_{gear}) F_n$$

$$F_{xB} = -F_{xA}$$

$$F_{yB} = -F_{yA}$$

(2)

(3)

(4)

(5)

(6)

(7)

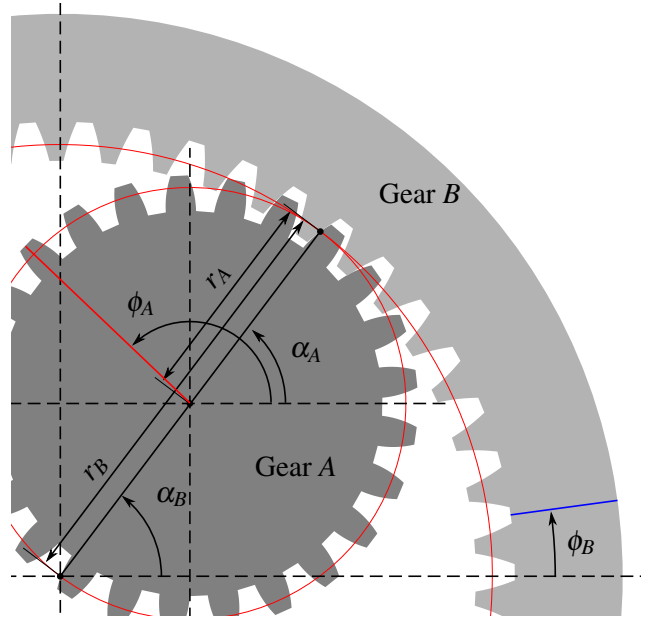


Figure 3: Schematic overview of two gearwheels in contact. The blue and red line are fixed markers on the gear wheels. In the figure $\omega_A > 0$ and Gear A drives Gear B.

2.2 Force and Moment balance of internal toothing

Just like in Section 2.1, the force and moment balance can be created by examining Figure 3 together with Figure 4:

$$\tau_A = F_n r_A \quad (8)$$

$$\tau_B = -F_n r_B \quad (9)$$

$$F_{xA} = -\sin(\phi_{gear}) F_n \quad (10)$$

$$F_{yA} = \cos(\phi_{gear}) F_n \quad (11)$$

$$F_{xB} = -F_{xA} \quad (12)$$

$$F_{yB} = -F_{yA} \quad (13)$$

3 Meshing distance

To keep track how the gear wheels move with respect to each other, the mesh distance x_{mesh} is introduced. This distance is defined as the distance the gear has traveled through the meshing point and can be calculated for both gear wheels. For the complete description of the mesh position the following assumption is postulated:

Assumption 1 The mesh contact position is on the direct connection between the center of gear A and B at a distance r_A from the center of A

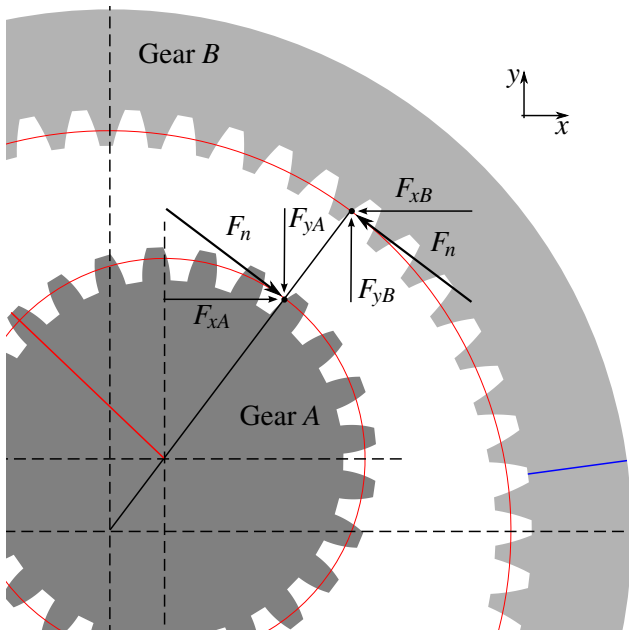


Figure 4: Free body diagram of the two gearwheels from Figure 3.

This assumption is valid for all cases in which the deformation of the tooth is small. In all engineering applications this must be the case for gearwheels under normal loading conditions.

3.1 Mesh Distance External Tothing

For external tothing the mesh distance can be calculated as follows using the geometry and definitions from Figure 1.

$$x_{mesh,A} = \phi_A r_A - \phi_{gear} r_A \quad (14)$$

$$x_{mesh,B} = -\phi_B r_B + \phi_{gear} r_B \quad (15)$$

From this equation it becomes clear that the mesh distance ($x_{mesh,A}$ or $x_{mesh,B}$) can be constant although the gear wheels are rotating. This is the case if $\phi_A = \phi_{gear}$ or $\phi_B = \phi_{gear}$. This is not only a theoretical implication; in e.g. bicycle gear hubs this is often the case.

The difference between the mesh positions is the elasticity of the gear contact:

$$\Delta_{AB} = x_{mesh,A} - x_{mesh,B} \quad (16)$$

Assuming the meshing position is always halfway the elastic deformation, together with using the equations 14 to 16 the mesh velocity is:

$$v_{mesh} = \dot{x}_{mesh,A} - \frac{\dot{\Delta}_{AB}}{2} \quad (17)$$

3.2 Mesh Distance Internal Tothing

The same analysis method can be applied to the internal tothing:

$$x_{mesh,A} = \phi_A r_A - \phi_{gear} r_A \quad (18)$$

$$x_{mesh,B} = \phi_B r_B - \phi_{gear} r_B \quad (19)$$

The difference between the mesh positions is as mentioned above the elasticity of the gear contact:

$$\Delta_{AB} = x_{mesh,A} - x_{mesh,B} \quad (20)$$

Assuming the meshing position is always halfway the elastic deformation, together with using the equations 18 to 20 the mesh velocity is:

$$v_{mesh} = \dot{x}_{mesh,A} - \frac{\dot{\Delta}_{AB}}{2} \quad (21)$$

4 Gear Wheel Coupling

The gear wheels A and B are coupled by a spring-damper combination. This yields:

$$F_n = \Delta_{ABC}(\phi_{gear}, \phi_A, \phi_B) + \dot{\Delta}_{AB} d(\phi_{gear}, \phi_A, \phi_B) \quad (22)$$

In this equation $c(\phi_{gear}, \phi_A, \phi_B)$ is the angle dependent spring constant and $d(\phi_{gear}, \phi_A, \phi_B)$ is the angle dependent damping constant.

4.1 Position Dependent Stiffness

The angle dependency can be used to simulate a non constant tooth stiffness. The total tooth stiffness is the combined stiffness of both gearwheels. Since the circumference of a gearwheel is periodic by definition, the following assumption can be postulated:

Assumption 2 *The position dependent stiffness and damping of a gearwheel can be described by a Fourier decomposition.*

One of the most basic forms of Assumption 2 is a single harmonic with zero phase offset that represents the tooth of the gear wheel. The stiffness over the circumference of a gearwheel can therefore be written as:

$$c_A(\gamma_A) = c_{const} + c_{\Delta,A} \sin(2\pi n_{tooth,A} \gamma_A) \quad (23)$$

$$c_B(\gamma_B) = c_{const} + c_{\Delta,B} \sin(2\pi n_{tooth,B} \gamma_B) \quad (24)$$

In this equation γ_A is the angle which describes the position of the material on the gear wheel. The stiffness at the contact position however, is dependent on which

part of the gearwheel is in contact. The local stiffness can be obtained for an external gear by using:

$$\gamma_A = \phi_A - \phi_{gear} \quad (25)$$

$$\gamma_B = -\phi_B + \phi_{gear} \quad (26)$$

Substituting Equations 25 and 26 into Equations 23 and 24 leads to the stiffness at the contact position.

$$c_{cont,A} = c_{const} + c_{\Delta,A} \sin(2\pi n_{tooth,A}(\phi_A - \phi_{gear})) \quad (27)$$

$$c_{cont,B} = c_{const} + c_{\Delta,b} \sin(2\pi n_{tooth,B}(-\phi_B + \phi_{gear})) \quad (28)$$

An internal gear configuration would yield:

$$\gamma_A = \phi_A - \phi_{gear} \quad (29)$$

$$\gamma_B = \phi_B - \phi_{gear} \quad (30)$$

leading to a contact stiffness of:

$$c_{cont,A} = c_{const} + c_{\Delta,A} \sin(2\pi n_{tooth,A}(\phi_A - \phi_{gear})) \quad (31)$$

$$c_{cont,B} = c_{const} + c_{\Delta,B} \sin(2\pi n_{tooth,B}(\phi_B - \phi_{gear})) \quad (32)$$

The overall stiffness can be calculated by putting both springs in series:

$$c = \left(\frac{1}{c_{cont,A}} + \frac{1}{c_{cont,B}} \right)^{-1} \quad (33)$$

5 Modelica Implementation

The presented gear contact model must be supplied by constraints in the x , y and ϕ direction (standard planar constraints). The Planar library from D. Zimmer [11] is used to supply these constraints. Features like (rotational) bearings, connection rods, inertias e.g. are all represented. The library will be used to create the total gearbox setup.

Implementation of the gear model in Modelica is straightforward using the sections above. The gear model is implemented with 2 planar interface connectors; each with 3 degrees of freedom (x, y, ϕ). These connectors are the connections to the gearwheels A and B . To sense the total revolution angle ϕ_{gear} ($\phi_{gear} \in \mathbb{R}$), the atan3 function is modified to supply a continuous and differentiable angle.

In Figure 5 the icons of the gear models are shown. No inertia's or constraints are included in the model.

Using the planar library, it is possible to create all kind of different gear configurations. Everything between

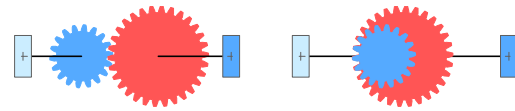


Figure 5: Modelica Icon of the inner and outer gear-wheel connections

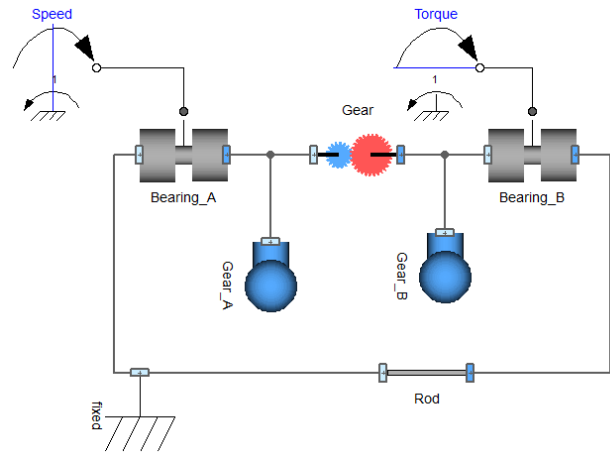


Figure 6: Spur Gear in Dymola

simple spur gears models (Figure 6 and 7) up to complex epicyclic gearing configurations (Figure 8 and 9) is easily generated. In these models, the gearbox models (Figure 5) are defined as described in this paper, all other components are components of the planar library (see [11]).

6 Simulation Results

6.1 Eigenfrequency Analysis

Using the Modelica LinearSystems2 library, it is possible to create a Bode-Diagram of a linear system. Since a linear spring and damper are used for the contact stiffness, it is possible to use this toolbox. Using an eigenfrequency analysis it is possible to check the

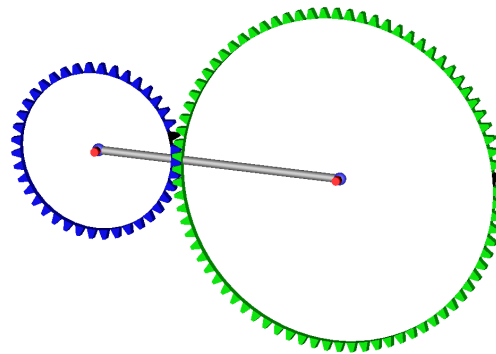


Figure 7: Spur Gear in Dymola

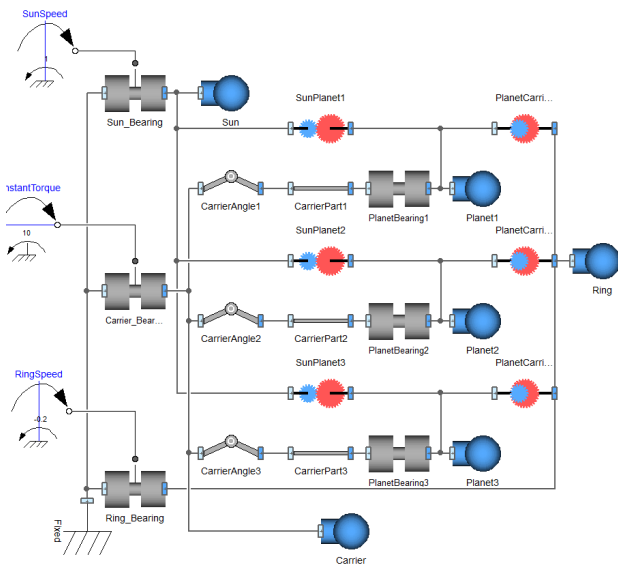


Figure 8: Epicyclic Gear

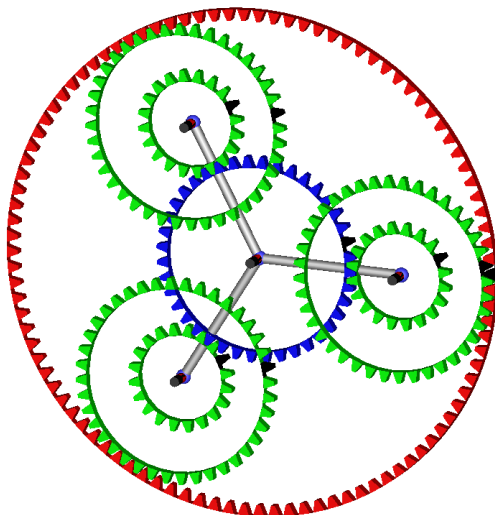


Figure 9: Epicyclic Gear

behavior of the models.

6.1.1 Spur Gear Analysis

A Single Input Single Output (SISO) system of a simple spur gear model (as shown in Figure 6) is generated by applying a torque input on gearwheel A, and using as output the angular position of gearwheel B. The Bode-Diagram of this system ¹ is shown in Figure 10. In the diagram a clear peak can be found

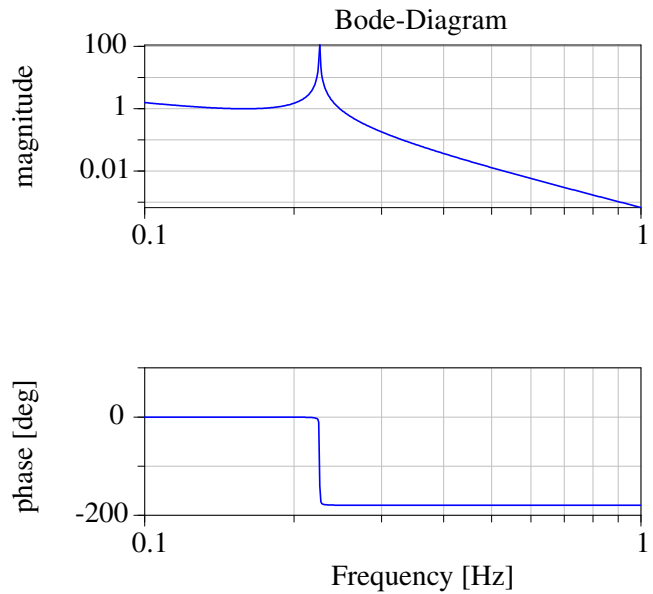


Figure 10: Bode-Diagram of the spur gear from Figure 6

at 0.225 Hz. This is exactly the expected frequency $\omega = \frac{\sqrt{k}}{2\pi} = \frac{\sqrt{\frac{2}{1}}}{2\pi} \cong 0.225$. The stiffness $k = 2\frac{N}{m}$ and mass $m = 1kg$ have to be used since the system is a symmetrical system using only one spring (see e.g. [2]). Lowering of the eigenfrequency due to damping can be neglected due to the low damping coefficient.

6.1.2 Epicyclic Gear Analysis

A SISO system is created by defining an input torque on the sun (middle (blue) gear in Figure 9), as output the angular position of the carrier (grey structure). The Ring (red) is fixed, thereby eliminating vibrations of the ring structure. Each small planet is coupled to the planet rotating on the same axis. All bodies have the following properties: Mass 1 kg, Inertia 1 kgm². All gear connections have a stiffness of $1\frac{N}{m}$, and a damping coefficient of $1e-3\frac{Ns}{m}$. The radius of the sun is

¹The bodies have a rotational inertia of 1 kgm², the spring constant of the gear is $1\frac{N}{m}$, and a damping coefficient is $1e-3\frac{Ns}{m}$. Both gearwheels have a radius of 1m.

1m, the connecting planet has a radius of 0.5m. The other gear part of the stepped planet has a radius of 1m. The ring has a diameter of 2.5m. Using this set up, a Bode-Diagram is made (see Figure 11). When

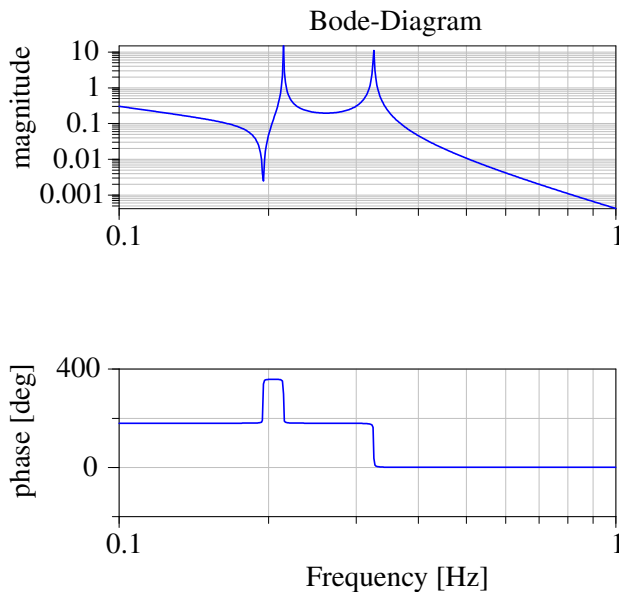


Figure 11: Bode-Diagram of the epicyclic gear from Figure 8

evaluating the Bode diagram, two peaks and a single dip can be found in the magnitude diagram. These features correspond to the 3 eigenfrequencies of the system. The fact that only 3 peaks can be found in the Bode diagram is due to the fact that the planets all have the same masses and stiffnesses. When the stiffness of one of the Sun-Planet gear connections is lowered to $0.5 \frac{N}{m}$, another peak and dip in the magnitude diagram occurs, since now one of the planets will swing in an other frequency as the others (see Figure 12).

6.2 Internal vibrations

In Section 4.1 the possibility of an internal excitation of the gear through varying stiffness is shown (to simulate gear mesh effects). A demonstration of this excitation is shown for a simple spur gear. Gear A is accelerated from $0 \frac{rad}{s}$ to $1 \frac{rad}{s}$ with a constant acceleration. A radius of 1m and 10 teeth for both gearwheels are assumed for this calculation. The constant tooth stiffness in the simulation is $1 \frac{N}{m}$, the stiffness ripple on both wheels is assumed to be 0.1%. Using a damping coefficient of $0.2 \frac{Ns}{m}$ this yields a lightly damped system with a damping ratio $\zeta \approx 0.071$. In Figure 13 the elastic deformation (Δ_{AB}) of the gear is shown.

In Figure 13 also shows that the system is excited by the internal mesh stiffness variation. The response of the system is the largest when the eigenfrequency of

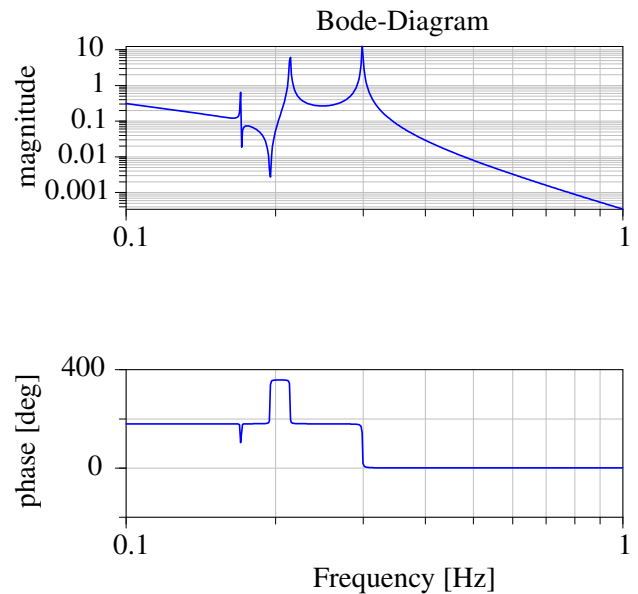


Figure 12: Bode-Diagram of the epicyclic gear from Figure 8 with reduced stiffness of one of the gear contacts.

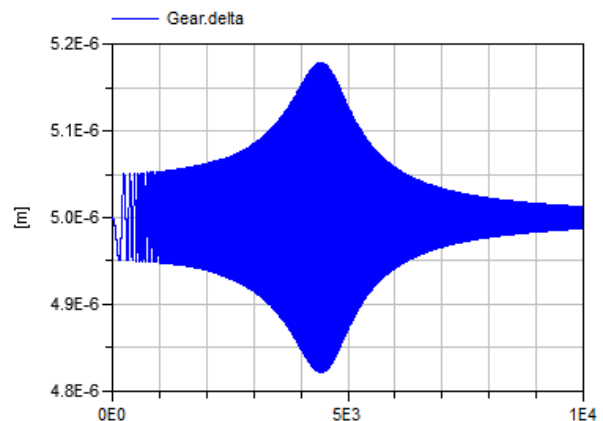


Figure 13: Time simulation of an elastic spur gear with increasing velocity.

the system approximates the excitation by the stiffness variation.

7 Conclusion

In this paper a model is presented to describe the contact between two gear wheels. Using an external planar library, it is possible to model arbitrary gear configurations ranging from simple spur gears up to complex epicyclic gear configurations. An option to simulate gear meshing effects by varying the stiffness of the gear contact is presented. The presented models make it possible to analyze complex gear configuration by means of time simulations as well as eigenfrequency

analyses. The presented simulation results show the power of the method, and illustrate the capability of the model.

Acknowledgements I thank Martin Otter for his help on Modelica related issues and Dirk Zimmer for his help with Modelica and his work on the Planar library.

References

- [1] HOWARD, I., JIA, S., AND WANG, J. The dynamic modelling of a spur gear in mesh including friction and a crack. *Mechanical Systems and Signal Processing* 15 (2001), 831–853.
- [2] KELLY, S. *Fundamentals of mechanical vibrations*. McGraw-Hill series in mechanical engineering. McGraw-Hill, 2000.
- [3] OTTEWILL, J. R., NEILD, S. A., AND WILSON, R. E. Intermittent gear rattle due to interactions between forcing and manufacturing errors. *Journal of Sound and Vibration* 321, 3-5 (2009), 913 – 935.
- [4] ÖZGÜVEN, N., AND HOUSER, D. Mathematical models used in gear dynamics - a review. *Journal of Sound and Vibration* 121, 3 (1988), 383–411.
- [5] PAREY, A., EL BADAQUI, M., GUILLET, F., AND TANDON, N. Dynamic modelling of spur gear pair and application of empirical mode decomposition-based statistical analysis for early detection of localized tooth defect. *Journal of Sound and Vibration* 294 (2006), 547–561.
- [6] PAREY, A., AND TANDON, N. Spur gear dynamic models including defects: A review. *The Shock and Vibration Digest* 35 (2003), 465–478.
- [7] PEDERSEN, R., SANTOS, I. F., AND HEDE, I. A. Advantages and drawbacks of applying periodic time-variant modal analysis to spur gear dynamics. *Mechanical Systems and Signal Processing* 24, 5 (2010), 1495 – 1508. Special Issue: Operational Modal Analysis.
- [8] PELCHEN, C., SCHWEIGER, C., AND OTTER, M. Modeling and simulating the efficiency of gearboxes and of planetary gearboxes. In *2nd International Modelica Conference* (2002), pp. 257–266.
- [9] SAWALHI, N., AND RANDALL, R. Simulating gear and bearing interactions in the presence of faults: Part i. the combined gear bearing dynamic model and the simulation of localised bearing faults. *Mechanical Systems and Signal Processing* 22, 8 (2008), 1924 – 1951.
- [10] VAN DER LINDEN, F., AND VAZQUES DE SOUZA SILVA, P. Modelling and simulating the efficiency and elasticity of gearboxes. In *Proceedings of the 7th International Modelica Conference, Como, Italy* (20–22 September 2009), pp. 270–277.
- [11] ZIMMER, D. A planar mechanical library for teaching modelica. In *review for the Proceedings of the 9th International Modelica Conference* (2012).

Revised and Improved Implementation of the Spur Involute Gear Dynamical Model

Ivan Kosenko* Ilya Gusev**

*Dorodnitsyn Computing Center of Russian Academy of Sciences, Department of Mechanics
Moscow, 119333, Russia

**Russian State University of Tourism and Service, Department of Natural and
Engineering Sciences, Cherkizovo-1, Moscow region, 141221, Russia

Abstract

An improved model having new, more realistic, properties is constructed with use of previously implemented approach for building up a model of the spur involute gear dynamics. First of all, an algorithm for contact tracking of cylindrical surfaces directed by involutes was rearranged. This algorithm is “simply” reduced to tracking the two involutes. A result is that common line normal to these contact curves always coincides with the line of action. This property permits obtaining direct simple formulae for contact computations.

A backlash in gearbox is also taken into account in the model under consideration. This means that a loss of contact between the teeth is possible as gearwheels rotate. This may then cause an appearance of a contact patch during the reversal. Furthermore, a dynamical reasons may force the mesh process to return to the former mode of the forward stroke and so fourth. All such scenarios for switching modes are implemented in the model in a unified way.

A time overlapping of contacts between teeth pairs is used to ensure the mesh reliability. This property is also implemented in the described dynamical model. New contact of the next pair of teeth arises and starts its motion along the line of action before the old contact leaves this line at the point of teeth disengagement.

Keywords: spur gear; involute; mesh properties; tracking algorithm; mesh ratio; multiple contact; backlash

1 Introduction

One can highlight two poles among all approaches to computer modeling and simulation of the gear dynamics. Computational algorithms of high accuracy are

relocated at one end of the corresponding scale. These algorithms take into account elasto-plastic properties of the material that the contacting bodies are made of, plus a variety of boundary conditions of different types [1]. Such high accuracy simulation models simultaneously require significant computational resources. One might point out different simplified models, see e. g. [2], on the other end of the scale. These models provide the highest efficiency.

The compromise model presented in [3] might be improved upon in a way so as to take into account essential properties of real gear: (a) backlash, (b) contact multiplicity. The latter property is always provided in real gears in order to prevent jamming in teeth. In addition, the contact tracking algorithm turned out to be simplifiable and simultaneously essentially acceleratable in the case of the involute mesh. For definiteness, we use the Johnson [4] model for the cylindrical bodies contact as was previously done [3] for the case of spur mesh.

2 Preliminaries

Using methodics [5, 6] previously developed for computer modeling of the rigid bodies 3D-motions let us consider planar motion for bodies of cylindrical shape, denoted as A and B in our case, in the plane orthogonal to generatrix of cylinders. We connect this plane with an additional body C , see Figure 1, an auxiliary frame O_Cxyz of coordinates is assumed to be rigidly connected with that latter body in a way such that cylindrical generatrix is always orthogonal to the axis O_Cz . One might express this latter requirement using the following geometrical conditions: $\mathbf{k}_\alpha = \mathbf{k}_C$ ($\alpha = A, B$), where \mathbf{k}_α ($\alpha = A, B, C$) are the unit vectors defining the axes $O_\alpha z$ connected with the bodies $\alpha = A, B$. To ensure the motion of the bodies in the plane parallel

to the plane O_Cxy let us require a fulfillment of yet two more algebraic conditions for the bodies A and B z -coordinates: $z_{O_A} = \text{const}$, $z_{O_B} = \text{const}$. All the coordinates are given with respect to (w. r. t.) the system O_Cxyz .

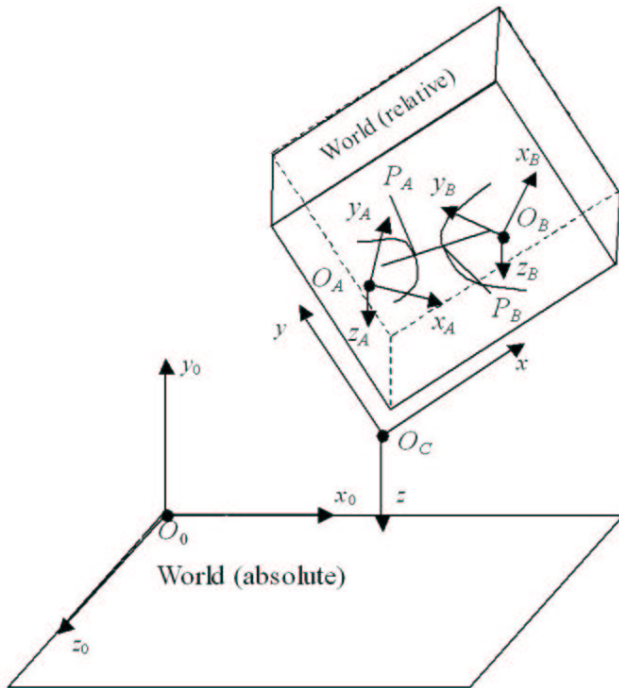


Figure 1: Coordinate systems for the model: (a) base frame of reference $O_0x_0y_0z_0$; (b) the gearbox housing coordinate system O_Cxyz ; (c) the pinion coordinate system $O_Ax_Ay_Az_A$; (d) the gear coordinate system $O_Bx_By_Bz_B$

One can easily implement algebraic equations enumerated above in implicit form. To fix the bodies A and B w. r. t. the body C one can use, for instance, constraints of the joint type [5, 6]. In this case the body C itself performs arbitrary 3D-motions being regarded as a convective motion w. r. t. certain inertial frame of reference. Thus calling the body C as the reduction-gear housing is quite natural, if the bodies A and B are models of gearwheels. After the reduction to the plane O_Cxy performed above building up a technique for the cylindrical bodies using 2D-geometry properties [3] is quite natural as well.

Note that all the bodies A, B, C in the model perform their 3D-motions according to the spatial dynamics of rigid body encapsulated in the corresponding base class. And relative cylindrical symmetry of bodies A and B w. r. t. the body C is kept due to the reaction forces between them. These forces are generated dynamically in an acausal mode due to kinematical constraints encapsulated in a contact class, rather a tem-

plate, being further constructed in this paper.

3 Account of the backlash

First of all, let us simplify and as a consequence essentially accelerate a performance of the previously implemented algorithm of the contact tracking for two involute surfaces of the teeth pair at the contact for the spur gear meshing. Such a simplification allows us building up the mesh model quite easily for the mesh ratio greater than one, and simultaneously accounting for the backlash.

As was found earlier [3] that in the case of the involute mesh the sought points P_A and P_B , see Figure 1, lying both on the perpendicular common for involutes of gearwheels teeth in vicinity of contact, are located simultaneously on the mesh line of action K_AK_B , see Figure 2. Evidently, the common perpendicular mentioned above also coincides with the line of action K_AK_B . Thus, from the geometric point of view the point P_A lies permanently in time on the intersection of the gearwheel tooth involute and the line K_AK_B . Similar statement takes place for the point P_B : it lies on the intersection of the gearwheel B tooth involute and the same line of action K_AK_B .

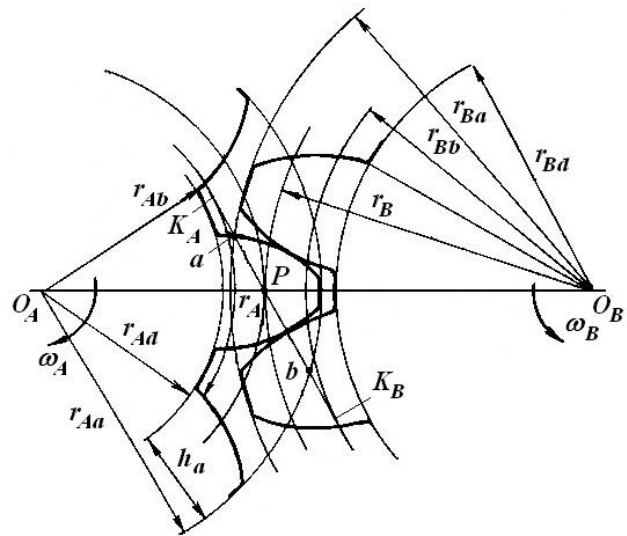


Figure 2: Gear mesh for forward stroke

Thus in case of involutes computing coordinates s_A and s_B of points P_A and P_B respectively on the strait line K_AK_B replacing a cumbersome algorithm using differential-algebraic equations is sufficient for contact tracking. One can compute coordinates s_A, s_B with an extremely simple procedure, see Figure 3.

Let the coordinates s_A, s_B denote the distances from

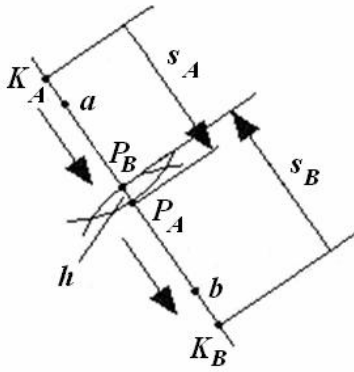


Figure 3: Contact tracking coordinates

the respective points K_A, K_B . We assume values of s_A, s_B at these source points being set to zero. Thus for $s_A + s_B \geq L = |K_A K_B|$ a contact takes place, and for $s_A + s_B < L$ the contact is absent. In the first case the depth h of the bodies mutual penetration is computed by the simple formula $h = L - s_A - s_B \leq 0$. Left arrows in Figure 3 show a direction in which the contact patch moves for the forward mode and as pinion rotates clockwise.

When computing the value h the pair of teeth being in contact is under analysis. In case of forward stroke we assume for definiteness that the wheel A, pinion, rotates clockwise while the wheel B, gear, supposed to rotate counterclockwise. The angles φ_A, φ_B of rotation of the bodies A, B respectively are defined by the axis $O_C x$ of the gearbox housing and by the axes beginning from bodies' points O_A, O_B and going through the points of their base circles where corresponding involutes "grow", see Figure 2.

Furthermore, if the wheel A, for definiteness, rotates such that the angle φ_A appears outside its admissible limits (being defined below) then the model generates an event corresponding to fulfilment of the condition $\varphi_A \notin [\varphi_{A \min}, \varphi_{A \max}]$. In such a case the values of angles $\varphi_A = \varphi_A^-, \varphi_B = \varphi_B^-$ are to be automatically corrected according to equations (we assume that contact of the forward stroke exists currently):

$$\varphi_A^+ = \varphi_A^- + m\Delta\varphi_A, \quad \varphi_B^+ = \varphi_B^- - m\Delta\varphi_B$$

for the case of $\varphi_A < \varphi_{A \min}$ and equations:

$$\varphi_A^+ = \varphi_A^- - m\Delta\varphi_A, \quad \varphi_B^+ = \varphi_B^- + m\Delta\varphi_B$$

for the case of $\varphi_A > \varphi_{A \max}$. Here $\Delta\varphi_A, \Delta\varphi_B$ are angular widths per one tooth of the wheels A and B respectively; m is the mesh multiplicity (the least integer greater than the mesh ratio). Note that the angles φ_A and φ_B are not exactly the bodies angles of rotation.

They are indeed the angles of rotation for wheels' teeth w. r. t. the axis $O_C x$. These teeth are supposed to lie currently in the zone of possible contact. This zone is defined by the condition $\varphi_A \in [\varphi_{A \min}, \varphi_{A \max}]$.

Formulae from above have to simply switch contact in the same sense as it was arranged in [3]. The following approximate rule is used: at the very same moment when the contact patch "instantly vanishes" behind an upper or lower limits of admissible segment $[\varphi_{A \min}, \varphi_{A \max}]$ this patch should appear immediately on the other end of the same segment. For simplicity the wheel A is considered as a "leading" object responsible for the event generation process.

Thus a current contact object of the model "jumps" to the next pair of teeth over $m - 1$ pairs being currently in contact if the object individual angle φ_A of tooth rotation goes out of its admissible limits. Recall that m is the mesh multiplicity, and in general we assume $m \geq 1$.

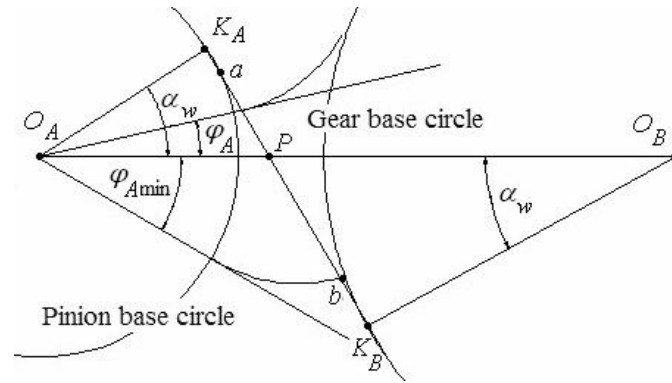
Limit values $\varphi_{A \min}, \varphi_{A \max}$ for angle of inclination of an involute at contact for the current pair of teeth are computed with natural restrictions being imposed on the contact area. Minimal value $\varphi_{A \min}$ corresponds to the final point b of contacting along the line of action for the case of forward stroke, see Figure 2. One can see easily that the value $\varphi_{A \min}$ is computed by the formula

$$\varphi_{A \min} = \alpha_w - \frac{|K_A a| + |ab|}{r_{Ab}}, \quad (1)$$

corresponding to the selection of points a and b where the contact process of starts and ends respectively. One might find details for such matching in [3]. Here α_w is the pressure angle, and r_{Ab} is the wheel A base circle radius. Equation (1) has a simple geometrical explanation, see Figure 4. Indeed, consider the pinion base circle. Its arc length from the point K_A downwards to the position corresponding to the angle $\varphi_{A \max}$ should be equal to the segment $[K_A, b]$ length according to the known involute property. This circumstance leads immediately to property (1). We recall that the point b on the gear mesh line of action defines the position where contact patch vanishes.

If the wheel A angular position $\varphi_A = \varphi_{A \min}$ corresponds to the instant for contact finishing then the angle $\varphi_A = \varphi_{A \max}$ has to correspond to this process beginning for the current pair of teeth. One easily sees that the assumption $\varphi_{A \max} = \varphi_{A \min} + m\Delta\varphi_A$ has to hold.

Similarly, obtaining formulae for computation of


 Figure 4: Limit angle φ_{Amin}

the points P_A, P_B coordinates s_A, s_B as

$$s_A = \begin{cases} r_{Ab}(\alpha_w - \varphi_A) & \text{for } \varphi_A < \alpha_w, \\ 0 & \text{for } \varphi_A \geq \alpha_w, \end{cases}$$

$$s_B = \begin{cases} r_{Bb}(\alpha_w + \pi - \varphi_B) & \text{for } \varphi_B < \pi + \alpha_w, \\ 0 & \text{for } \varphi_B \geq \pi + \alpha_w, \end{cases}$$

where r_{Bb} is the wheel B base circle radius, is not very difficult.

One has to provide additional contact (between wheels A and B) objects to take into account all the possible contacts of teeth pairs if the mesh ratio is greater than one. To simulate the gearbox forward stroke one has to provide generally m instances of such contact objects.

Furthermore, clearly, if contact of teeth in the forward stroke vanishes then it is almost evident that contact of reversal arises. This latter arises between the teeth pair closest to contact lost before and being located through the tooth trough on the involutes of the teeth sides previously unused in the forward stroke mode.

To simulate the reversal one has to use line of action derived from the line of Figure 2 by mirroring it w. r. t. the axis connecting points O_A and O_B . All the mesh geometric properties considered for the forward stroke are mirrored for the case of reversal. In particular, coordinates s_A and s_B for this case have expressions

$$s_A = \begin{cases} r_{Ab}(\alpha_w + \varphi_A) & \text{for } \varphi_A > -\alpha_w, \\ 0 & \text{for } \varphi_A \leq -\alpha_w, \end{cases}$$

$$s_B = \begin{cases} r_{Bb}(\alpha_w - \pi + \varphi_B) & \text{for } \varphi_B > \pi - \alpha_w, \\ 0 & \text{for } \varphi_B \leq \pi - \alpha_w. \end{cases}$$

Note that what we have meant under “the forward stroke” or “reversal” is not a kinematical property whether to rotate clockwise or counterclockwise but it is a dynamical property switching into work/contact between driving/driven surfaces of teeth. Thus, we

will see the forward stroke in cases of the pinion A clockwise accelerated and counterclockwise decelerated rotation. Similarly, reversal takes place in general if pinion A accelerates when rotating counterclockwise and decelerates simultaneously rotating clockwise. Simplifying formulations let us call the rotation with line of action shown in Figure 2 as the forward stroke. Likewise, the rotation with line of action mirrored w. r. t. the axis $O_A O_B$ of Figure 2 be called as reversal. The reversal requires correct switching between pairs of teeth, as well as, it was implemented for the forward stroke.

When contacting in reversal mode switching of the teeth pairs takes place if the contact patch leaves the segment $a'b'$ of line of action $K'_A K'_B$, see Figure 5, or by the point a' or through the point b' . For that one has to apply the same relations as above replacing the segment $[\varphi_{Amin}, \varphi_{Amax}]$ of admissible values for the angle φ_A by the segment $[\varphi'_{Amin}, \varphi'_{Amax}]$ for the angle φ'_A such that $\varphi'_{Amin} = -\varphi_{Amax}$, $\varphi'_{Amax} = -\varphi_{Amin}$.

4 Case of multiple contact

Previously mesh ratio was supposed equal to one in the simplified model of the gear mesh [3]. This means that exactly at the moment of contact loss at the point b new contact at the point a arises. Such an arrangement leads frequently to a low reliability of a gearbox as well-known however in practice, mostly due to jamming caused by manufacturing errors. Due to this reason ensuring a reliable gearbox work one provides overlapping for time intervals of contacts in teeth pairs. Namely, new contact at the point a arises earlier than the current contact vanishes at the point b .

Let us return to the example being analysed in [3] where a virtual setup for computational experimenting was constructed, see Figure 6 and also Figure 1 for ge-

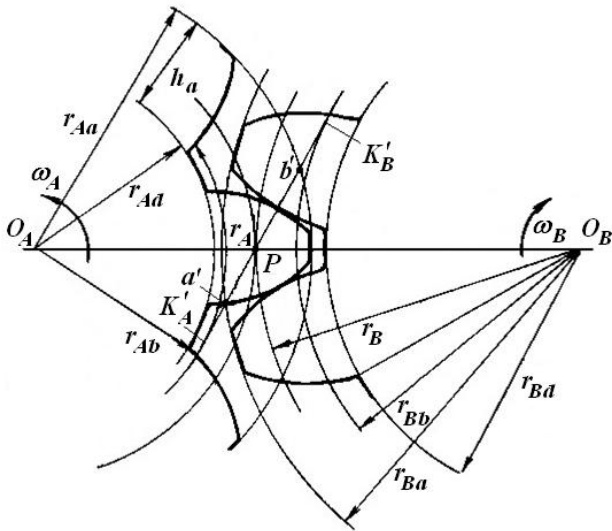


Figure 5: Gearmesh for reversal mode

ometry of the whole construct. This setup consists of two gearwheels: pinion A and gear B . For simplicity we assume the gearbox housing C fixed w. r. t. the base body of a whole multibody system. Furthermore, origin O_C of an inertial frame O_Cxyz of reference coincides with the pinion A center where the revolute joint which connects bodies A and C is located. The gear B center locates on the horizontal axis O_Cx . There exists the second revolute joint connecting the body B and auxiliary slider S . The slider S in turn may freely slip w. r. t. the body C along the axis O_Cx . This slipping however is decelerated by a spring of very large stiffness. The spring connects the bodies C and S thus providing a compliance between the bodies B and C through the intermediate slider S . This compliance has direction along the line O_AO_B connecting the wheels centers and coinciding with the axis O_Cx . Such a construct prevents static indefiniteness in the model for the case of the rigid point-contact in the gearmesh of the wheels A and B .

We define in the model the following independent parameters:

- $z_A = 20$ is number of the pinion teeth;
- $z_B = 30$ is number of the gear teeth;
- $r_A = 0.2\text{m}$ is the pinion pitch circle radius.

Other (dependent) geometric parameters are computed as follows

- $n = z_B/z_A$ is the transmission ratio;
- $r_B = nr_A$ is the gear pitch circle radius;

- $\Delta\gamma_A = 2\pi/z_A$, $\Delta\gamma_B = 2\pi/z_B$ are the pitch angles of the pinion and gear.

For further definition of the gear mesh choosing the pressure angle value is important. This value has to satisfy the condition $\alpha_w > \alpha_{w\text{inf}}$, where $\alpha_{w\text{inf}} = \inf \alpha_w$ is the lower bound for all possible pressure angles which are admissible by parameters selected above. One can compute this bound according to the formula

$$\alpha_{w\text{inf}} = \arctan \frac{2\pi}{z_A(1+n)}.$$

The lower bound obtained above is a simple consequence of the mesh natural condition

$$|\overrightarrow{K_A K_B}| > |\overrightarrow{ab}|.$$

For definiteness let us choose the value

$$\alpha_w = 2.8\alpha_{w\text{inf}}.$$

Furthermore, with the help of the pressure angle value and the value of the transmission ratio we can compute all the geometric parameters needed shown in Figure 2. Firstly of all one can obtain radii of base circles as

$$r_{\alpha b} = r_{\alpha} \cos \alpha_w \quad (\alpha = A, B).$$

Then one can compute full length of the line of action in the following way

$$|\overrightarrow{K_A K_B}| = r_A(1+n) \sin \alpha_w.$$

At the same time, the length of any segment of contact $[a, b] \subset (K_A K_B)$ along this line is exactly the length of the base circle arc corresponding to the pitch angle $\Delta\gamma_A$ or $\Delta\gamma_B$ for any wheel of the gearbox. Thus we have

$$|\overrightarrow{ab}| = r_{\alpha b} \Delta\gamma_{\alpha} \quad (\alpha = A, B).$$

One easily computes the distance between wheels centers as $L = r_A + r_B$. For computing initial conditions in the model performing additional calculations is necessary. Suppose for definiteness that the coordinate system $O_Cx_Cy_Cz_C$ has its origin at the point O_A of the pinion A center: $O_C = O_A$, so that these points initial absolute coordinates coincide. Thus

$$\mathbf{r}_{O_C} = \mathbf{r}_{O_A} = (0, 0, 0)^T,$$

and the initial position of the gear center is defined by

$$\mathbf{r}_{O_B} = (L, 0, 0)^T.$$

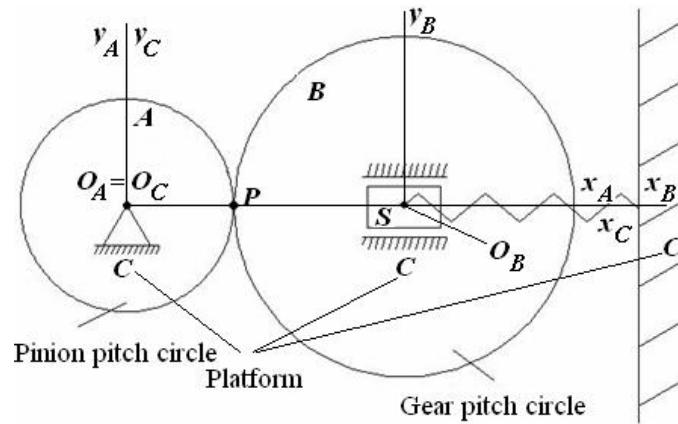


Figure 6: Virtual setup for computational experiments

Initial positions of the points K_A and K_B are computed by vector formulae

$$\begin{aligned} \mathbf{r}_{K_A} &= \mathbf{r}_{O_A} + r_{Ab} (\cos \alpha_w, \sin \alpha_w, 0)^T, \\ \mathbf{r}_{K_B} &= \mathbf{r}_{O_B} - r_{Bb} (\cos \alpha_w, \sin \alpha_w, 0)^T \end{aligned}$$

being deduced easily. Furthermore, a directing vector for the line of action is defined as $\overrightarrow{K_A K_B} = \mathbf{r}_{K_B} - \mathbf{r}_{K_A}$. So that the contact starting point a initial position may be defined as

$$\mathbf{r}_a = \mathbf{r}_{K_A} + \frac{1}{2} \left(\frac{|\overrightarrow{K_A K_B}| - |\overrightarrow{ab}|}{|\overrightarrow{K_A K_B}|} \right) \overrightarrow{K_A K_B},$$

and the initial position for the point b of contact finishing as

$$\mathbf{r}_b = \mathbf{r}_a + \frac{|\overrightarrow{ab}|}{|\overrightarrow{K_A K_B}|} \overrightarrow{K_A K_B}.$$

Let us take into account that the distance between the points a and O_B is exactly equal to the addendum circle radius r_{Ba} for the wheel B , and the initial distance from the point b to O_A is equal to the addendum circle radius r_{Aa} for the wheel A . Namely

$$r_{Aa} = |\mathbf{r}_b - \mathbf{r}_{O_A}|, \quad r_{Ba} = |\mathbf{r}_a - \mathbf{r}_{O_B}|.$$

To ensure overlapping of the mesh cycles for wheels with transmission ratio $n = 3/2$ let us consider the case with $z_A = 22$ and $z_B = 33$ providing the same transmission ratio. Note that the real angular widths $\Delta\varphi_A = \pi/11$ and $\Delta\varphi_B = 2\pi/33$ for teeth become less than their nominal, pitch, widths $\Delta\gamma_A = \pi/10$ and $\Delta\gamma_B = 2\pi/30$.

Simultaneous coexistence of two contacts in the model obtained requires, both in the forward stroke and in reversal, the use of four contact objects in the mesh computer model — two for the forward stroke

plus two for reversal. Visual model of the experimental setup is represented in Figure 7. Here `Contactf1` and `Contactf2` are objects for the forward stroke, and `Contactb1` and `Contactb2` are ones for reversal. Thus as a result models for the pinion, the left wheel object `LeftWheel`, and gear, the right wheel object `RightWheel`, each has four input ports for information about wrenches arising at patches of an elastic contacts. Object `Platform` simulates dynamics of the base body (absolute world), gearbox housing C (relative world), having a predefined motion, resting in our case. Thus two mentioned worlds coincide for the model of Figure 7.

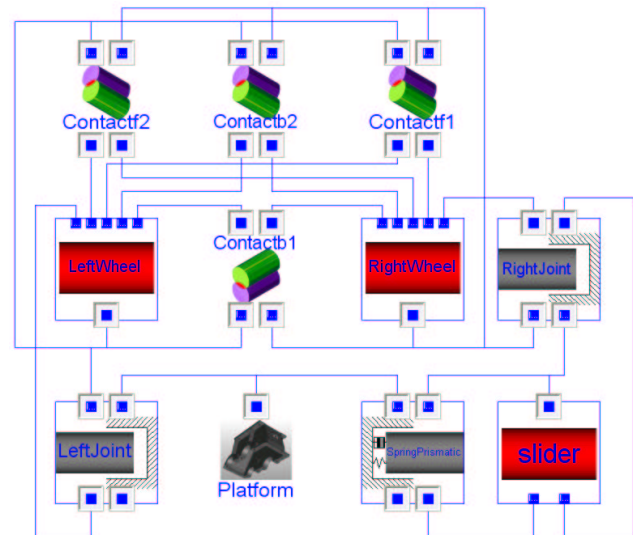


Figure 7: The testbench visual model

Note that each contact object mentioned above works virtually independently. Coordination of their on/off switching is achieved using proper and correct selection of initial conditions for the state variables inside contact objects. In case of our dynamical example

these conditions are defined in the following way.

Angular velocities of wheels are assumed to be zero. For definiteness we also suppose that the body A axis $O_A x_A$ goes through the involute root point lying on the pinion base circle. This involute defines exactly the tooth surface, and initially it goes through the point a thus starting a contact. Likewise, the axis $O_B x_B$ of the wheel B initially goes through the gear B involute root point.

At initial instant of time one pair of the wheels A and B teeth is supposed to have contact at the point a , see Figure 2, with a zero depth of mutual penetration. For definiteness this contact is supposed to be in the forward stroke mode. One can easily conclude from Figure 2 that in initial position of teeth for angles of inclination φ_A , φ_B of radius vectors for the involute root points, lying on the base circles, the equations

$$\begin{aligned}\varphi_{A0}^{f1} &= \text{atan2}(a_y, a_x) - \theta_A, \\ \varphi_{B0}^{f1} &= \text{atan2}(a_y - y_{C_B}, a_x - x_{C_B}) - \theta_B,\end{aligned}\quad (2)$$

where θ_A , θ_B are the point a polar angles on teeth involutes, hold. These angles are computed in the form

$$\begin{aligned}\theta_A &= \frac{\sqrt{|\mathbf{a}|^2 - r_{Ab}^2}}{r_{Ab}} - \arccos\left(\frac{r_{Ab}}{|\mathbf{a}|}\right), \\ \theta_B &= \frac{\sqrt{|\mathbf{a} - \mathbf{r}_{C_B}|^2 - r_{Bb}^2}}{r_{Bb}} - \arccos\left(\frac{r_{Bb}}{|\mathbf{a} - \mathbf{r}_{C_B}|}\right).\end{aligned}$$

Thus wheels initial angles of inclination in the object `Contactf1` are defined by formulae (2). Furthermore, for the mesh ratio being greater than one if one pair of teeth starts contact at the point a then another neighbour pair being ahead of the previous one will have a contact somewhere on the segment $[a, b]$. This latter contact is supposed to be defined in the object `Contactf2`. Initial values of auxiliary angles φ_A , φ_B defining the angles of involute rotation for the forward stroke (or reversal) and being respectively in segments $[\varphi_{A \min}, \varphi_{A \max}]$, $[\varphi_{B \min}, \varphi_{B \max}]$ are to be distanced from the angles of the object `Contactf1` exactly by the tooth angular width (which is smaller than the angular pitch of the gear mesh under simulation). Namely, the following formulae

$$\varphi_{A0}^{f2} = \varphi_{A0}^{f1} - \Delta\varphi_A, \quad \varphi_{B0}^{f2} = \varphi_{B0}^{f1} - \Delta\varphi_B, \quad (3)$$

are to be satisfied.

An initial data selection for the objects `Contactb1`, `Contactb2` servicing the reversal is not so evident. Indeed, involutive surfaces of the teeth pair being tracked by the object `Contactb1`

will be situated on the same teeth as the surfaces being tracked by the object `Contactf2`. The only difference is that they should be relocated on other sides of the teeth mentioned, see Figure 2. So from geometrical point of view contact of reversal being tracked by the object `Contactb1` on initial stage of motion should be located between the contacts of the forward stroke being tracked by the objects `Contactf1` and `Contactf2`.

Note that for a particular tooth the radius vectors of the involute root points (lying on the base circles) of its two sides rotate w. r. t. each other exactly by an angular width of one tooth without accounting for the tooth trough. Note that all angular widths are to be counted along the arcs of the base circle. Denote these angular widths of teeth by $t_{A \text{wid}}$, $t_{B \text{wid}}$. Then one can compute initial data in the object `Contactb1` by the formulae

$$\varphi_{A0}^{b1} = \varphi_{A0}^{f2} + t_{A \text{wid}}, \quad \varphi_{B0}^{b1} = \varphi_{B0}^{f1} + t_{B \text{wid}}.$$

Let us remark here that really at the initial instant of the computational experiment the objects `Contactb1`, `Contactb2` generate zero-valued wrenches of contact forces. All this is due to the contacts absence for reversal mode though objects `Contactb1`, `Contactb2` always continue to track the points P_A , P_B inside each of them.

The object `Contactb2` of the second contact for the reverse mode has the following initial data for the involute angles of inclination (rather angles of inclination of their root points radius vectors for the case $m = 2$)

$$\varphi_{A0}^{b2} = \begin{cases} c & \text{for } c > \varphi_{A \min}, \\ d & \text{for } c \leq \varphi_{A \min}, \end{cases}$$

where $c = \varphi_{A0}^{f1} - 2\Delta\varphi_A + t_{A \text{wid}}$, $d = \varphi_{A0}^{f1} + t_{A \text{wid}}$ and

$$\varphi_{B0}^{b2} = \begin{cases} q & \text{for } c > \varphi_{A \min}, \\ r & \text{for } c \leq \varphi_{A \min}, \end{cases}$$

where $q = \varphi_{B0}^{f1} + \Delta\varphi_B + t_{B \text{wid}}$, $r = \varphi_{B0}^{f1} - \Delta\varphi_B + t_{B \text{wid}}$.

In the latter equations we take into account the fact that for the case of the mesh multiplicity for the reversal mode there exist several possibilities, two in our example, of contact implementations along the line of action $K'_A K'_B$, see Figure 5.

Ensuring the initial data selection from above in the objects of contact we thus automatically provide correct switching of modes of contact inside the objects and correct tracking for involutes contact switching in the process of wheels rotation. The built up mesh

model provides a possibility to simulate motions of any type in the gearbox with any combination for contact between teeth. This model enables us able to construct effectively the gearboxes virtual prototypes of any complexity for the case of the spur involute gear.

5 Behavioral Model of Contact Object

Let us return to the gearbox visual model presented in Figure 7. It has been built with the help of earlier proposed [5, 6] technologies for constructing the physically-oriented models. For each physically implemented contact of the model there exists one object of visual model, see Figure 7. Meanwhile, from the functional viewpoint there is no difference how contacts of specific type, Nos. 1 and 2 for the forward stroke and Nos. 1 and 2 for reversal, are redistributed over an array of unified contact objects. Thus, the same class code is able “to play a role” of contact of any type within the spur involute gear model. In virtue of the circumstances outlined above organizing an array for all four contact objects in virtual model is reasonable. There should also be an array of four connectors reserved for transmitting data of wrenches from contact objects to objects of bodies, the wheels *LeftWheel*, *RightWheel* in Figure 7. In this case corresponding wrench ports are to be really arrays of ports [7] in objects *LeftWheel* and *RightWheel*.

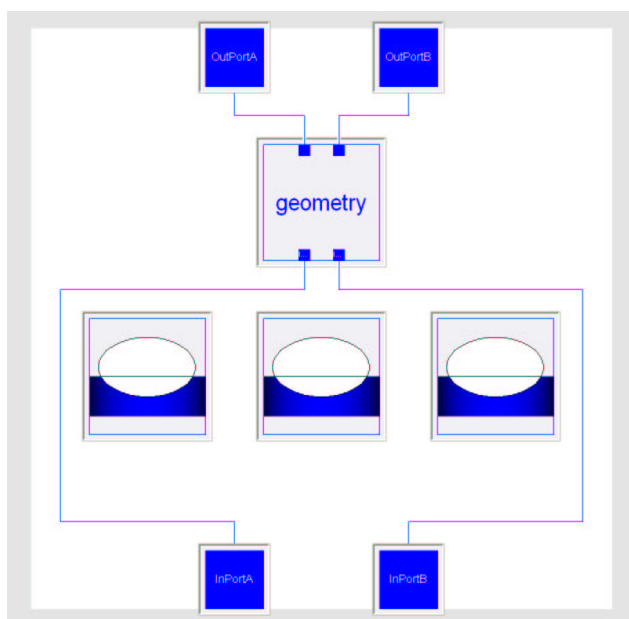


Figure 8: Base template for contact model

Note that according to the approach previously im-

plemented in [8] contact objects have a class being a template which has four class parameters responsible for implementation of: (a) geometry of surfaces at contact; (b) model of normal elastic contact forces; (c) model of normal viscous forces; (d) model of tangent (usually friction) forces. Visual representation of base template with empty sockets for the above four parameters see in Figure 8. The final derived class is shown in Figure 9 with mentioned sockets filled, actually redeclared, respectively by the following model parameters: (a) involute cylinder – involute cylinder; (b) the Johnson contact model for cylindrical bodies; (c) non-linear normal viscous model; (d) simplified Coulomb model of friction for tangent forces.

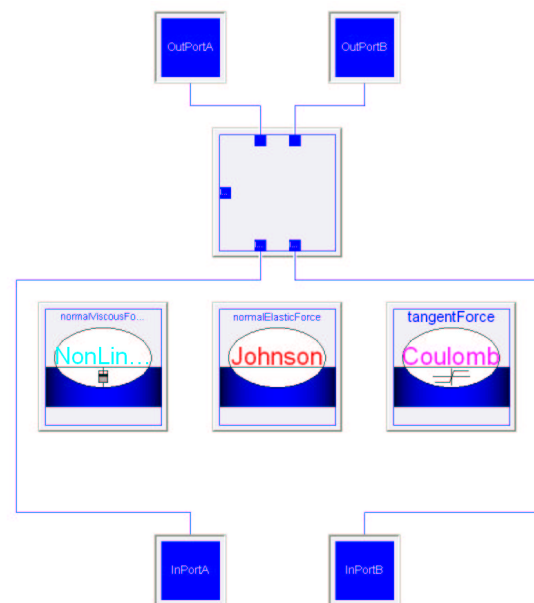


Figure 9: Final derived class for contact model of the example

The angles φ_A , φ_B of the wheels relative rotation are fundamental properties of the contact model under consideration. The angle φ_A is defined clearly in Figure 4. The angle φ_B has a similar sense for the wheel *B*. These angles remain always bounded throughout simulation: $\varphi_\alpha \in [\varphi_{\alpha\min}, \varphi_{\alpha\max}]$ ($\alpha = A, B$). Simultaneously for derivatives the equations $\dot{\varphi}_\alpha(t) \equiv \dot{\psi}_\alpha(t)$ are fulfilled almost everywhere for $t \in [t_0, t_1]$ where t_0 is the starting instant of the simulation process, t_1 is the instant of the simulation finish. Here the values ψ_α ($\alpha = A, B$) are assumed angles of the gearwheels rotation w. r. t. the gearbox housing. The variables ψ_α ($\alpha = A, B$) are defined by dynamical equations of the model. In general the angle $\psi_\alpha(t)$ may grow or decrease infinitely. At the same time the angles $\varphi_\alpha(t)$ always remain bounded. The property described above

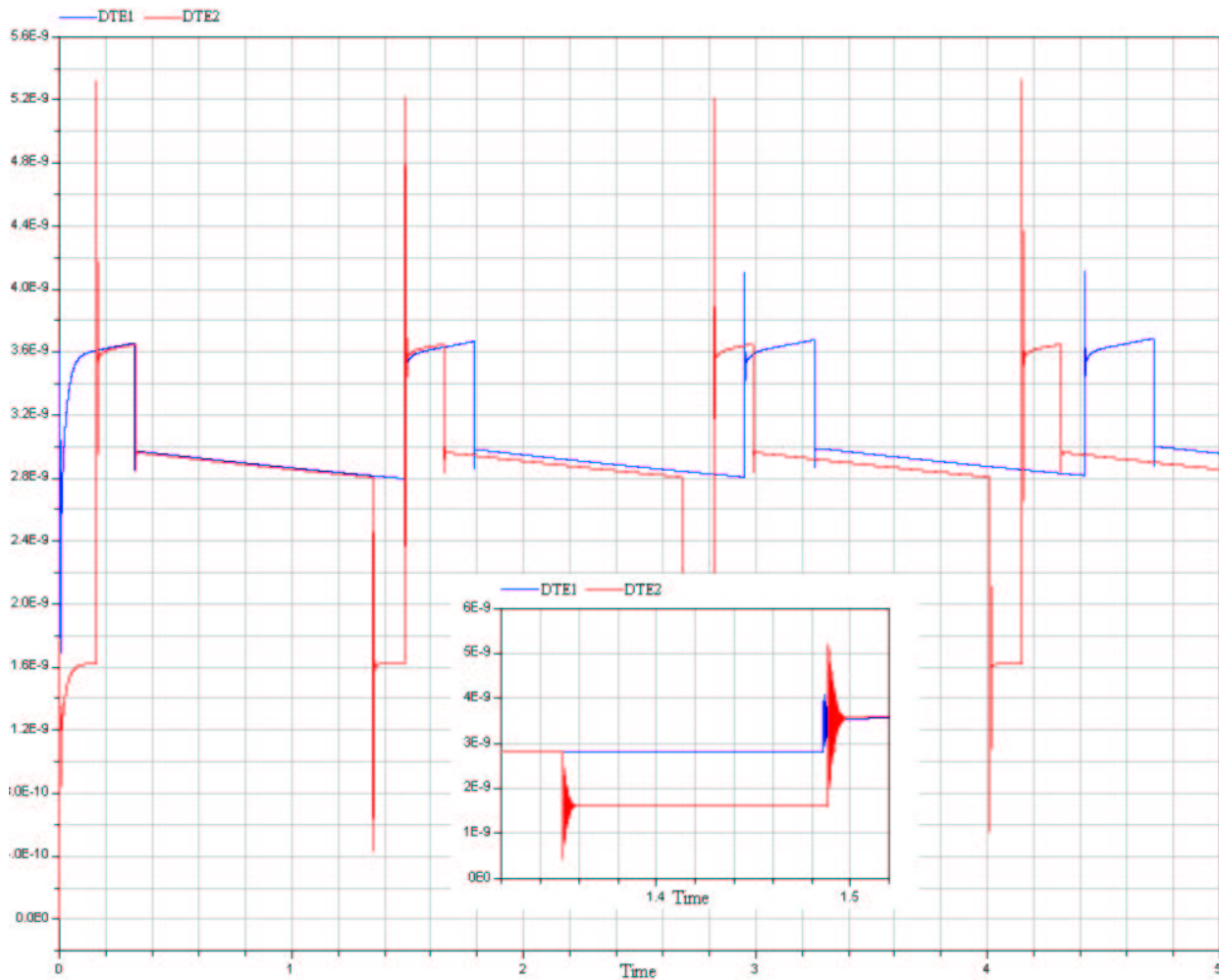


Figure 10: Dynamical transmission error

is implemented using the technique of event processing.

Namely, in usual mode if $\varphi_\alpha(t) \in [\varphi_{\alpha\min}, \varphi_{\alpha\max}]$ then we assume that the value $\varphi_\alpha(t)$ satisfies the differential equation

$$\frac{d\varphi_\alpha}{dt} = \frac{d\psi_\alpha}{dt} \quad (\alpha = A, B), \quad (4)$$

where the derivative is supposed to be expressed depending on state variables of the problem. When one of the events: $\varphi_\alpha(t) < \varphi_{\alpha\min}$ or $\varphi_\alpha(t) > \varphi_{\alpha\max}$ occurs then the initial condition of Cauchy problem is corrected immediately in the corresponding differential equation of the system (4) at the instant $t = t_*$ of the event according to the formula

$$\varphi_\alpha(t_*) = \begin{cases} \varphi_{\alpha\max} & \text{for } \varphi_\alpha(t_*-) = \varphi_{\alpha\min} \text{ and } \\ & \dot{\varphi}_\alpha(t_*-) < 0, \\ \varphi_{\alpha\min} & \text{for } \varphi_\alpha(t_*-) = \varphi_{\alpha\max} \text{ and } \\ & \dot{\varphi}_\alpha(t_*-) > 0. \end{cases}$$

The technique of event processing outlined above provides us with the correct model for simulating

physical switching for teeth at contact for the gearbox model simulation.

6 Numerical experiments

To verify an improved model of the gearbox numerical experiments were performed similar to those of the work [3]. Graphs for the dynamical transmission error (DTE) and value of the normal elastic force at contact were under verification. For DTE the current model clarifies the time dependence tracking as it has been done in [9, 10] instants for increasing/decreasing of the contact multiplicity. One can observe splashes of the value under observation, DTE here, as it was observed also in [9, 10] at these moments, see graph of DTE in Figure 10.

The DTE graph for the previous model from [3] is represented here by the blue curve (variable DTE1): the mesh multiplicity is equal to one, and then the teeth contacts overlapping is absent. The red curve (variable

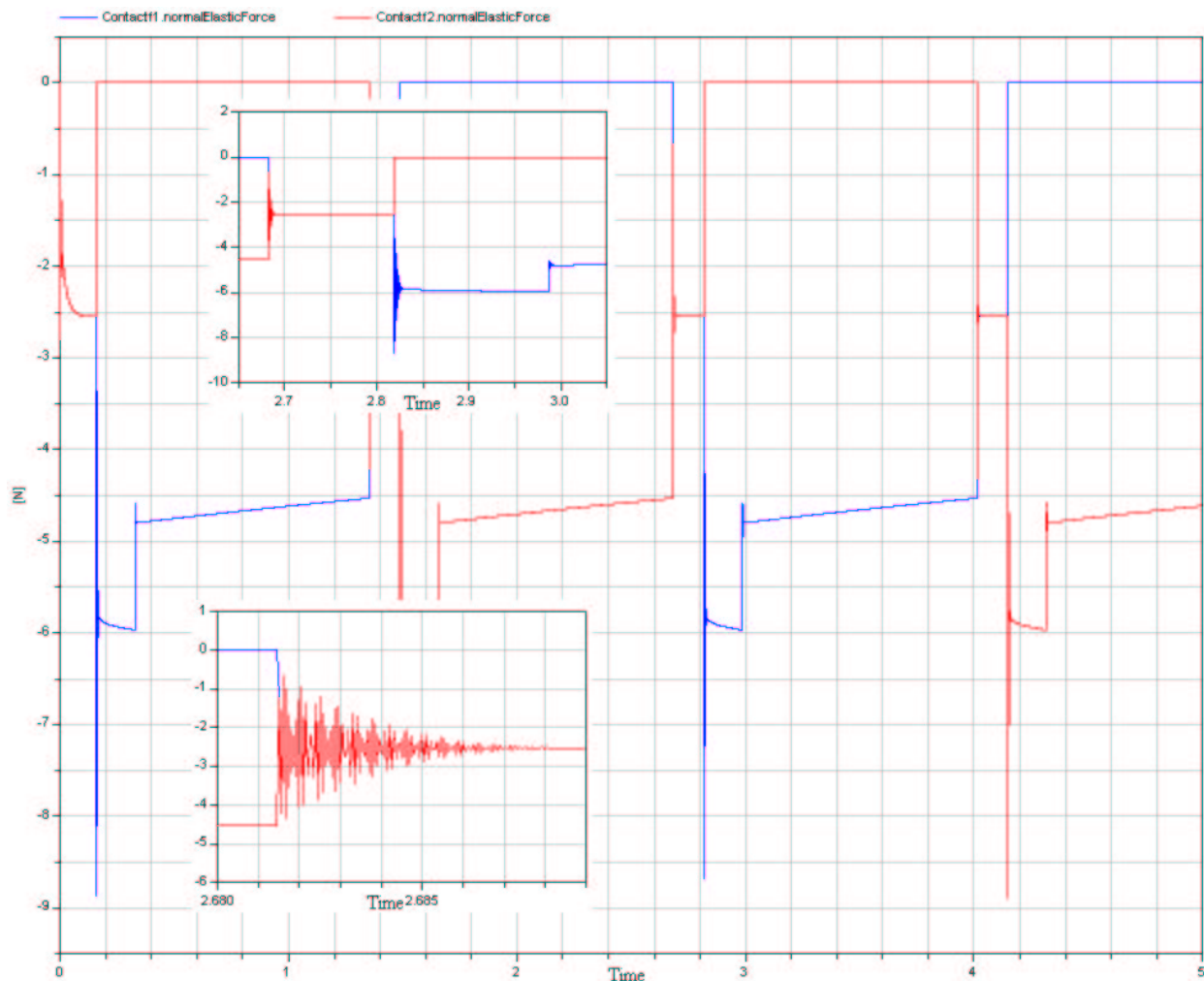


Figure 11: Normal forces at contacts

DTE2) corresponds to the case of the mesh multiplicity being equal to two. In this case there are time intervals for two simultaneously existing contacts, see subplot zoomed in in Figure 10. Interval of two contacts begins with the left splash. Then DTE instantly decreases because total contact stiffness increases with two contacts. Right splashes correspond to the instant of the old contact vanishing. Then only new contact remains. Anyway, in the case under description there exists an overlapping in time for contacts. And yet another observation: as one can also see from the graph an effect of overlapping causes a systematic shift of the mesh cycle. Indeed, total duration of each individual contact mesh cycle remains the same as it was in case of unit multiplicity. At the same time the periodicity in meshing for the case of two contacts becomes shorter by the duration of overlapping interval.

When exploring a behavior of the normal elastic force we can observe yet another interesting phenomenon. Usually following an engineering tradition

one applies the so-called restricted formulation of dynamical problem with multiple contacts: for computing the normal elastic forces at each contact one simply divides the total elastic force by the number of contacts being currently in action, see for instance [9, 10]. In our current approach, on the contrary, we compute normal elastic force at each contact individually from dynamics and with the use of the Johnson contact model. So one may say that we have implemented so-to-speak unrestricted problem for teeth contact of the spur involute gear. The normal contact force behavior along the mesh cycle is shown in Figure 11.

In the Figure 11 one exhibits the time dependence for elastic forces being generated in two different contact objects of the model. One assumes in the case under analysis that the constant accelerating torque acts upon the pinion *A*, while the gear *B* is under the torque of viscous resistance which is proportional to the value of the wheel angular velocity. The graph presents us yet another interesting, though quite nat-

ural, observation: an engineering approach which has been described above is indirectly verified by the exact dynamical model with an unrestricted contact model – values of normal elastic forces for contacts concurrently existing turned out to be almost identical. This observation takes place at least for the case of large contact stiffness corresponding to the steel our gearwheels are supposed to be made of.

7 Conclusions

Comparing results obtained in [3] with results of current work we can highlight the following properties:

1. The model is capable of simulating both the forward stroke and reversal of the gearbox taking into account a possibility of backlash between teeth.
2. The model is capable of simulating the involute mesh with multiple contacts.
3. The most effective, for the case of involute as tooth profile, contact tracking algorithm is implemented in the model. All this is due to differential or algebraic equations were excluded from the model, and only direct computations were in use.
4. To ensure an accuracy of the model of contact the most suitable implementation turned out to be an array of contact objects. Coordination of their behavior is provided by proper selection of initial conditions for the object variables.

8 Acknowledgements

The paper was prepared with partial support of Russian Foundation for Basic Research, projects: 11-01-00354-a, 12-01-00536-a, 12-08-00637-a.

References

- [1] Ziegler, P., Eberhard, P. Simulation of Geartrains with an Elastic Model. In: Proceedings of Multibody Dynamics 2011. An ECCOMAS Thematic Conference, Université catholique de Louvain, Brussels, Belgium, July 4–7, 2011, 11 pp. ISBN 978-2-8052-0116-5.
- [2] Pelchen, C., Schweiger, C., Otter, M. Modeling and Simulating the Efficiency of Gearboxes and of Planetary Gearboxes. In: Proceedings of 2nd International Modelica Conference, Deutsches Zentrum für Luft- und Raumfahrt e. V. (DLR), Oberpfaffenhofen, Germany, March 18–19, 2002, pp. 257–266.
- [3] Kosenko, I., Gusev I. Implementation of the spur involute gear model on Modelica. In: Proceedings of the 8th International Modelica Conference, Auditorium Centre of the Technische Universität Dresden, Germany, March 20–22, 2011, pp. 315–328.
- [4] Johnson, K. L.: Contact Mechanics. Cambridge: Cambridge University Press, 2001.
- [5] Kosenko, I. I., Loginova, M. S., Obratsov, Ya. P., Stavrovskaya, M. S. Multibody Systems Dynamics: Modelica Implementation and Bond Graph Representation. In: Proceedings of the 5th International Modelica Conference, arsenal research, Vienna, Austria, September 4–5, 2006, pp. 213–223.
- [6] Kosenko, I. I. Physically Oriented Approach to Construct Multibody System Dynamics Models Using Modelica Language. In: Proceedings of Multibody 2007, Multibody Dynamics 2007. An ECCOMAS Thematic Conference, Politecnico di Milano, Milano, Italy, June 25–28, 2007.
- [7] Fritzson, P. Principles of Object-Oriented Modeling and Simulation with Modelica 2.1. Piscataway, NJ: IEEE Press, 2004.
- [8] Kosenko, I., Aleksandrov, E., Implementation of the Contensou–Erismann Model of Friction in Frame of the Hertz Contact Problem on Modelica. In: Proceedings of the 7th International Modelica Conference, Como, Italy, 20–22 September 2009. Francesco Casella, editor. Linköping University Electronic Press, 2009. ISBN 978-91-7393-513-5. Linköping Electronic Conference Proceedings, ISSN:1650-3740. DOI: 10.3384/ecp0943, pp. 288–298.
- [9] Vaishya, M., Singh, R. Sliding Friction–induced Non–Linearity and Parametric Effects in Gear Dynamics. *Journal of Sound and Vibration*, Vol. 248, pp. 671–694, 2001.
- [10] Vaishya, M., Singh R. Strategies for modeling friction in gear dynamics. *Journal of Mechanical Design*, Vol. 125, pp. 383–393, 2003.

Accessing External Data on Local Media and Remote Servers Using a Highly Optimized File Reader Library

Dipl.-Ing.
Jörg Rädler

Dipl.-Ing. Manuel
Ljubijankic

Prof. Dr.-Ing. Christoph
Nytsch-Geusen

M.Sc.Dipl.-Ing.(Fh)
Jörg Huber

Berlin University of the Arts / Universität der Künste Berlin (UdK)

Hardenbergstrasse 33, 10623 Berlin, Germany

jraedler@udk-berlin.de

manuel@udk-berlin.de

nytsch@udk-berlin.de

jhuber@udk-berlin.de

Abstract

ncDataReader2 [1] is an open-source solution for the efficient interpolating access to external data sets. The library of C-functions can be used with different applications and works well with Modelica. Data sets can be easily accessed as continuous functions using different interpolation and extrapolation methods. The application range covers reading generated or measured data, the integration of simulation results from Modelica or other systems and the validation, parametrization and optimization of models using external data. Data sources may be local files or remote servers. Using the netCDF file format [2], the DAP network protocol [3] and different optimization approaches the data access can be surprisingly fast, even for large remote files with many variables containing millions of values.

1 Introduction

Getting external data into a simulation model is an important task for a lot of applications: buildings and energy plants are exposed to weather factors, complex models need to be validated with measured values and some simulations require results from other simulation runs.

The access conditions can vary significantly. A dense grid of data can be interpolated in small or in large intervals, and so can a wide grid. A large dataset may be evaluated only in one point to compute initial values or interpolated a million times, moving backward, forward or randomly on the x-axis. For some of these conditions and small amounts of data the *Table*-like classes of the Modelica Standard Library are a good choice, but for different application scenarios the ncDataReader2 offers some real advantages:

- very fast, even with large amounts of data

- load on demand (only needed data is read and processed)
- low memory consumption (adjustable, suitable for embedded simulations)
- clever caching mechanisms, tunable for different access characteristics
- different interpolation and extrapolation methods
- offset and scaling of values for unit conversion and memory-efficient storage
- API¹ (ANSI C) and data files work the same way in Modelica systems and other applications
- data can be accessed locally or with a highly efficient network protocol (DAP)

Although used mainly for 1D data sets the library includes basic support for variables depending on two dimensions (scattered 3D-points)². This paper will focus on the 1D functions.

1.1 History and Development

The development of the file reader library started more than 10 years ago as a tool for the DAE simulation system SMILE. Until now it was constantly improved and tested with SMILE [5], ANSYS CFX [6], the Modelica systems OpenModelica and Dymola and with proprietary applications.

ncDataReader2 is open-source software, everybody is invited to use and improve it under the terms of the GNU LGPL [7].

¹ API - Application Programmers Interface: the data and functions available for the programmer
² using the csa library from [4]

1.2 netCDF – the file format

netCDF is a binary file format³ and a program library developed for large amounts of multi-dimensional geoscientific data. The big advantage over other formats is the ability to access pieces of data without reading whole data sets or even whole files. netCDF files are self-describing and may contain structured data of different dimensions. This makes a very good format to archive numerical data and a perfect base for a file reader used in DAE simulations.

1.3 Interpolation and Extrapolation

ncDataReader2 includes the interpolation methods Akima (most used), linear, discrete and smoothed steps⁴ (see figure 1). Akima interpolation is a cubic method that gives smooth results (C1-continuity) without the tendency to overshoot. In contrast to classical cubic spline interpolation the points have only local influence, which perfectly complements the local access in netCDF. To get an interpolated value only some of the neighbouring points have to be read and processed after the search for the matching interval.

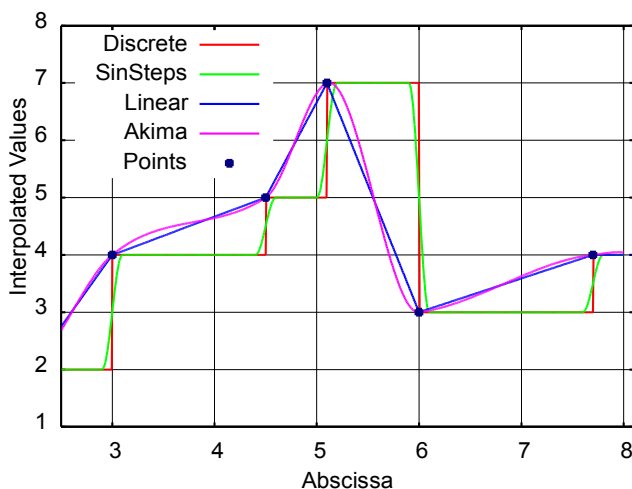


Fig 1: Interpolation methods

Extrapolation methods are implemented as periodic or natural (depending on the interpolation method).

³ Recent versions of the netCDF format are based on the HDF5 file format which is now used in MATLAB and many other applications.

⁴ Adjustable continuous approximation of discrete characteristics with C1-continuity, using linear parts and $\sin()$ -functions. Strictly speaking this is no real interpolation since the points are often not hit.

1.4 Tuning and Optimization

Variables may be fully loaded at initialization time, loaded in chunks of a specific size or as single values on demand. Three different caches may be enabled and changed in size:

- a lookup cache stores results of the interval search,
- a parameter cache holds the parameters of the linear or cubic function of an interval (both for successive requests of nearby values) and
- a value cache contains the last computed values (for successive requests of the same values).

The effect of these optimizations strongly depends on the access characteristics but may give a speedup factor of 100 and more in some cases.

The methods for interpolation and extrapolation as well as all parameters regarding loading, scaling and caches are preset to reasonable default values. All settings may be adjusted using attributes in the netCDF file or with the C API (full API only, see below).

Performance Example

The effect of clever using the optimization methods can be demonstrated with the example BigFile contained in the library. A data file with a size of 840 MB contains 10 variables each with 10 million random values. A Modelica class integrates the interpolated values of two of these variables (Akima method) over a sub-range of the abscissa. The result of the 500000 time steps is written to the result file.

With OpenModelica using default settings on a common PC⁵ the program finishes in about 5 seconds! This includes the complete run with initialization, data reading, more than one million interpolations, numerical integration and writing of the results. Some other approaches would need much longer just to load the data file.

This performance is achieved by a combination of the lookup and parameter caches and by loading the data in chunks on demand.

⁵ OpenModelica 1.8.1, Ubuntu Linux, Intel Core2 Duo E7200@2.53GHz, 4GB RAM

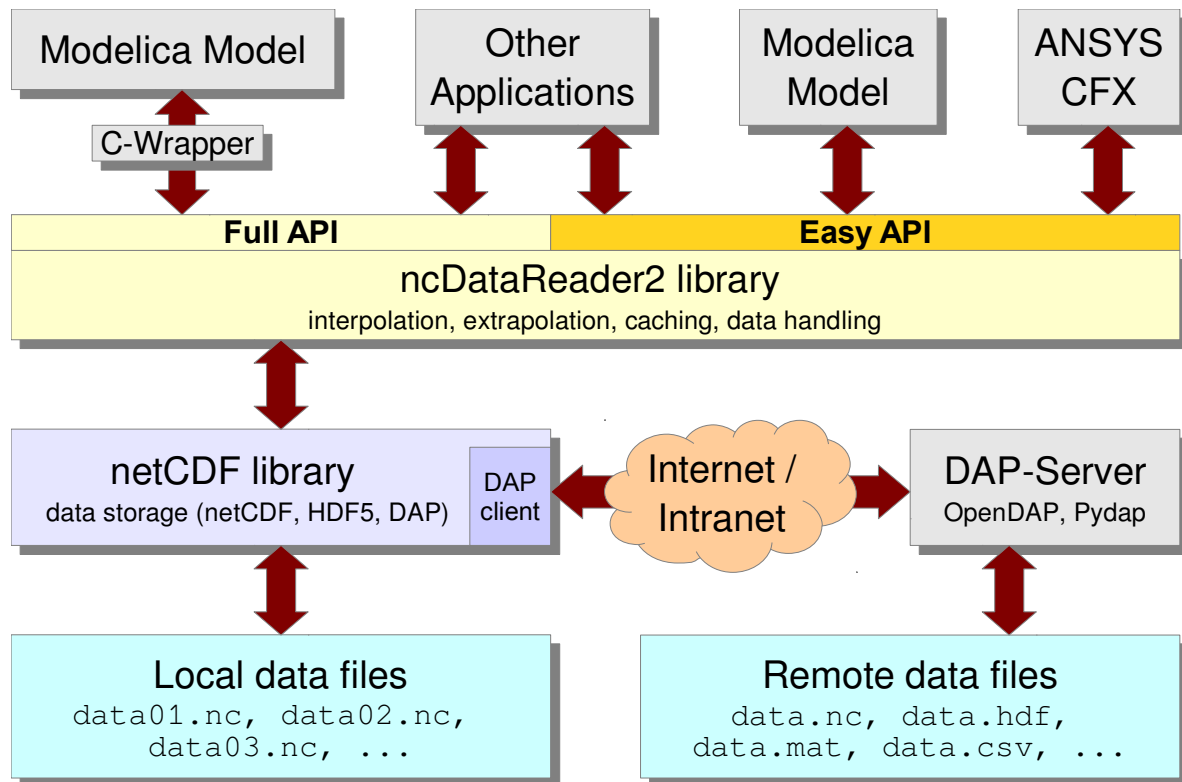


Fig 2: Different ways of using *ncDataReader2* with Modelica and other applications

2 Modelica API

ncDataReader2 offers a full API and a so called easy API. The latter limits the configuration options and requires compliance to some restrictions, but it can be used in Modelica without writing C code. The full API is slightly faster and offers access to all options, but uses data types not available in Modelica. Therefore it requires adapted external functions and a bit more programming.

Both methods require the prior installation of the libraries *ncDataReader2* and *netCDF* (which may depend on other libraries). The names of the files actually needed depend on the operating system, simulation software and compiler⁶.

Most Linux distributions already contain the required packages for *netCDF*. For Windows precompiled files (.dll, .lib, .h) are provided by the developers.

The structure of the different APIs and libraries is shown in figure 2.

⁶ The search for files by the compiler on Windows systems may be confusing. Copy all files to the current working directory if in doubt.

2.1 Easy API

The Modelica package *NcDataReader2* contains definitions of all functions of the easy API. A very basic example demonstrates the usage:

```
model NcEasyTest
  import nc = NcDataReader2.Functions;
  parameter String fn = "etest.nc";
  Real t;
  Real a = nc.ncEasyGetAttributeDouble(
    fn, "", "start_value");
  equation
    t = nc.ncEasyGet1D(fn, "v1", time);
    der(a) = t;
end NcEasyTest;
```

Two functions of this package are used here:

- `ncEasyGetAttributeDouble` reads a scalar attribute to initialize `a`. The first and third arguments are the names of the file and the attribute. The second argument may be a variable name or empty (to use a global file attribute). Similar functions exist for attributes of integer and string types.
- `ncEasyGet1D` returns the interpolated value of the variable `v1` at the point `time`. A similar function for 3D-points exists.

At the first call to `ncEasyGet1D` the file is opened, the abscissa and dimensions of the variable are determined, optional attributes are evaluated and internal data structures are created and stored for later use. Subsequent calls with the same file name and variable name reuse these structures. Different variables depending on different abscissas in different files can be used at the same time.

2.2 Full API

The full API can only be used in C, not in Modelica. To utilize this API wrapper functions are required to hide the complexity from Modelica. The function definitions are split up in two parts:

- a) A C-file which defines wrapper functions with simple interfaces (arguments and return values) to be used in Modelica. Inside these functions the full API may be used. A common case is to have one function with initialization code and one small function for each variable to return the interpolated values. This can usually be done within a couple of lines. All settings and options of the library may be changed in this file.
- b) A Modelica file containing mappings of the C-functions to Modelica functions (usually 1:1). This includes the number and types of arguments and return values.

Although these steps are quite simple, an example would be too big to show here. Please have a look at the example in *NcDataReader2.Examples.FullAPI*.

3 Preparation of Files

Converting data into the netCDF format may be the hardest task for users without prior knowledge of netCDF. There exists no general graphical tool for this job, but besides command line tools for the conversion of an ASCII-based format there are libraries for all major programming languages (C, C++, Java, Python, Perl, Octave, MATLAB, ...) and platforms.

A new project [8] from Microsoft Research provides a .NET-API, a graphical data viewer, command line tools and a plug-in for MS EXCEL to read and manipulate netCDF files on Windows systems.

The favourite method of the authors is scripting in Python. A lot of file formats can be read with

Python, and consistency checks and unit conversions may be included in a script. The conversion of a simple CSV file can be done within 7 lines of Python code. This method works on all platforms.

When using a DAP server the conversion may be omitted entirely for some file formats.

4 Data Server with DAP

DAP is a protocol for the transport of multidimensional gridded data over networks. It is based on HTTP, but allows the request and the transport of specific parts of a file in different formats. DAP servers are able to serve requests like “*send me the values 1500...2000 of the variable 'temperature' in the file 'data.mat' converted to CSV format*”. Clients can browse and request data with ease and efficiency. Data formats are converted on demand by the software (supported formats depend on the actual implementation).

Since newer versions of the netCDF library implement the client side of the DAP protocol, a DAP server can be used with `ncDataReader2` like a local file, just by replacing the file name with an URI.

For Modelica users this combination offers a lot of options. External data used by simulations can be hosted on different servers worldwide. During simulation, only the actually needed parts are transferred to the simulation system. This ensures the up-to-dateness and the consistency of data across simulations and allows the cooperation of different institutions without sending complete files.

4.1 DAP server at the UdK

A new server at the Berlin University of the Arts (UdK) was installed for this purpose. It provides different kinds of data to a research group:

- Weather files for different locations worldwide, generated with METEONORM [9] and converted to netCDF (see 5.1).
- Data from the monitoring of a photovoltaic system located at the UdK main building. The data is read from the monitoring hardware and stored in netCDF files in regular intervals (see 5.3).

- Results from different simulations of the research group.

The server runs on common PC hardware using Linux, Apache and the Pydap [10] software.

5 Current Applications in Research

5.1 Reading Weather Files

Data sets with weather parameters were the first application for the data reader and still are most used. Thermal building simulation and simulations of solar systems require reliable information about the environmental conditions as functions of time. These conditions include:

- temperature, pressure and moisture of the air,
- wind speed and direction,
- direct and diffuse radiation, cloud cover.

For the evaluation of the radiation on different surfaces the position of the sun is needed, which can be calculated from the latitude, longitude and time zone of the location.

All this information can be easily stored in a netCDF file. Over the years some conventions regarding the file structure, the units and the names of variables and attributes have evolved and proved to be useful.

All this data is read and processed by a Modelica class (*DataWeatherNetCDF*). With the file name or URI as a parameter of this class the complete environmental conditions of a location may be set and changed. The contained quantities and some derived quantities are available as continuous functions of time. Common problems like negative radiation values caused by the cubic interpolation are handled. For different oriented surfaces the radiation values will be converted by a helper class (*RadiationTransformer*). All these classes are now part of the new Modelica library *BuildingSystems* [11] which is developed by the authors.

Generating Files

The files may be created from different data sources. The authors mainly used weather information from the TRY/Testreferenzjahr [12] and data sets generated with the application METEONORM. The latter let the user define own ASCII-based export formats, which can be easily converted to netCDF by

a Python script. Our script now includes consistency checks, unit conversions, preparation for periodic extrapolation and much more.

With this method a library of weather files for different locations is built and expanded. The files reside on the DAP server (see 4.1) and are accessible by the whole research group. Data for new locations or new conditions can be generated and made available within minutes.

The time grid of most data files is equally spaced in hourly steps covering one year, but the software stack (DAP, netCDF, ncDataReader2, Modelica classes) works perfectly with different and variable steps and in other scales.

5.2 Loose Coupling of ANSYS CFX with Dymola

A research project at the UdK covers the co-simulation of a solar thermal plant. For pre-studies of a use case the ncDataReader2 is used for loose coupling. It reads the results of a Modelica simulation into the boundary conditions of a CFD⁷ simulation with ANSYS CFX.

The main components of the plant are:

- an evacuated tube collector,
- a hot water storage and
- an external plate heat exchanger, transferring the produced thermal energy from the solar loop to the storage loop.

By using a two-point-controller the solar pump and the storage pump are simultaneously switched on. The system is modelled with Modelica, most components are based on the Modelica library *BuildingSystems* for thermo-hydraulic network simulation. The weather data is provided by the technology described in the previous section.

The storage model (marked in Fig. 3) can be either implemented in Modelica (1D) or in CFX (3D). The co-simulation of Modelica and CFX is described in [13]. It gives more accurate results regarding the details of the storage, but it runs much slower than the pure Modelica simulation.

Additionally stand-alone CFX simulations of the storage component were needed in the project. One

⁷ Computational Fluid Dynamics

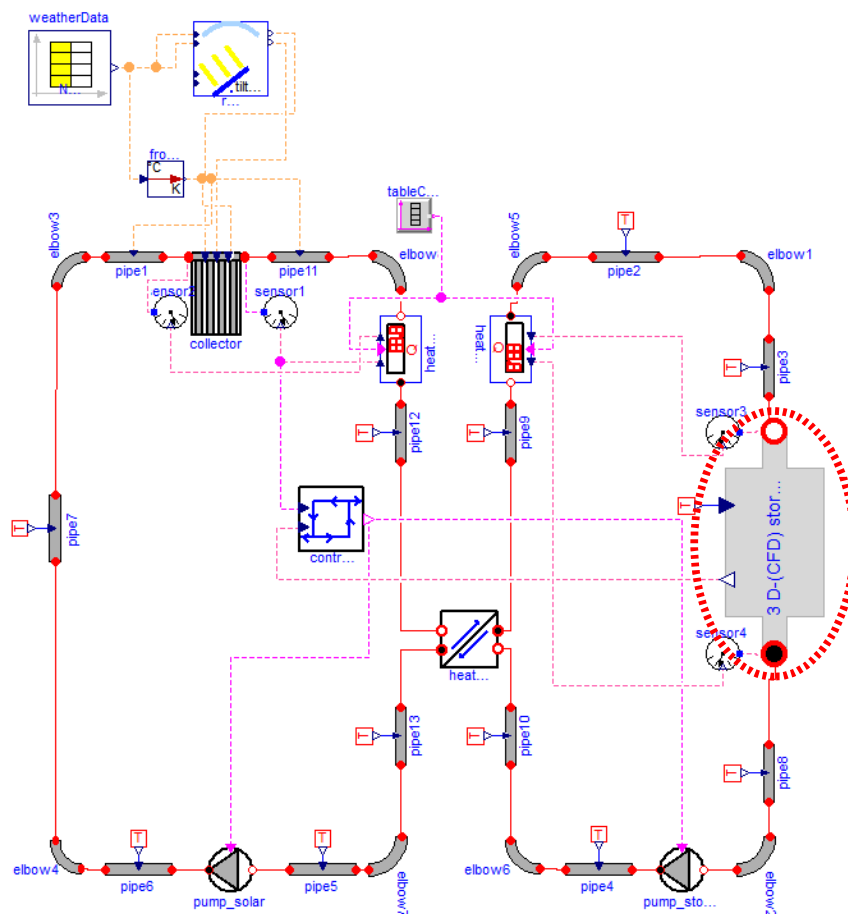


Fig 3: Modelica view of the co-simulation model of the solar thermal plant

of the questions to answer was the best resolution of the grid for the 3D model under typical conditions in the solar system. A high resolution will slow down the simulations, a wide grid will not reach the desired accuracy. A complete co-simulation model proved to be too slow to study this point in detail.

For a transient stand-alone CFX simulation of the thermal storage some boundary conditions are necessary to describe the installation situation. It would be possible to generate the time-dependant conditions with C-functions emulating the behaviour of the whole system, but the effort for this would be enormous. At this point it's much more comfortable to use the results from the system simulation with the simple storage model implemented in pure Modelica.

Dymola creates a result files in MATLAB format during the simulation. The structure of this file is quite complex, but can be read and converted with different tools. One is the Python package DyMat [14] which directly exports variables to different formats including netCDF.

The CFD model needs values for three quantities:

- inlet mass flow,
- inlet temperature,
- outlet pressure.

The time series for these variables can now be saved in a netCDF file.

ANSYS-CFX provides an API to implement dynamic conditions as Fortran functions. Since it is possible (but tricky) to call C-functions from this Fortran-API, ncDataReader2 can be used from this API with a small wrapper file to provide the values as interpolated functions of time.

The complete workflow is now:

- a) Run simulations in Modelica using the pure Modelica storage model.
- b) Convert the required results from the mat-file to netCDF format using DyMat.
- c) Run CFX simulation of the complex storage model, reading boundary conditions from the netCDF file using ncDataReader2 and a Fortran wrapper.

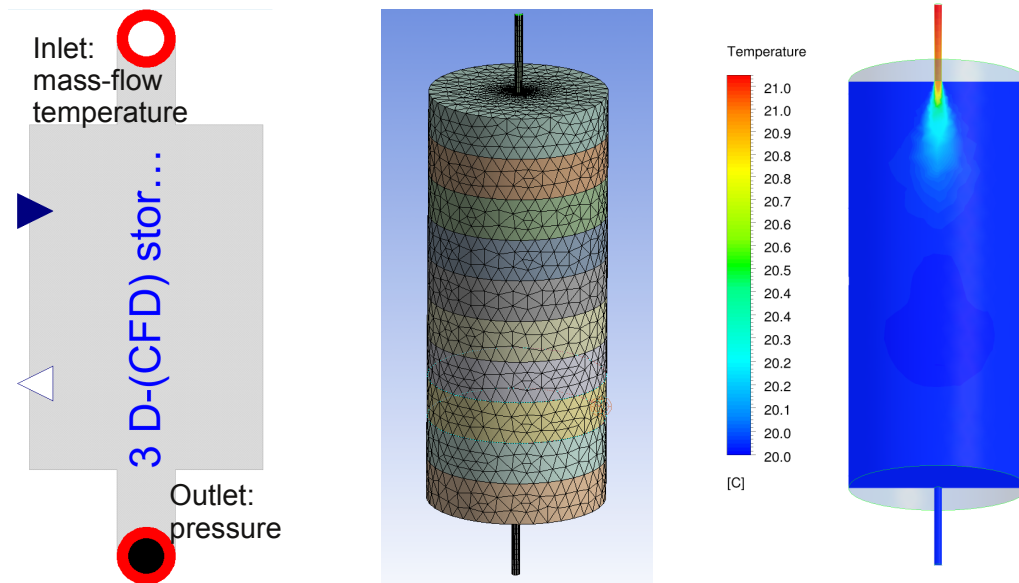


Fig 4: Storage model: a) Modelica connection component for the 3D model, b) grid of the CFX model, c) example of a temperature field using boundary conditions from the Modelica simulation

With this technology a stand-alone ANSYS-CFX simulation for the thermal storage can be started with dynamic adapted boundary conditions after each time step. Thus the CFX model is embedded in the same environment as the Modelica storage model in the system simulation for the solar thermal plant before.

This made it possible to research and tune the CFX model with respect to grid resolution and other parameters under typical conditions. Similar conditions appear in the real co-simulation which is the main topic of the research project [15].

5.3 Parametrization of the Model of a Photovoltaic Plant

The ncDataReader2 was used for the integration of measured data from a real photovoltaic (PV) system into a simulation model of the system. The complete field has an electrical power output (peak) of 15.5 kW and is located on the roof of the UdK Berlin main building. The measured values such as air and module temperature, solar irradiation, electrical output are used as climate boundary conditions of the Modelica system model and as comparison values for the model validation (see Fig. 5 and 6).

The Modelica model was configured by the use of the *BuildingSystems* library. One of the three strings of the photovoltaic field was modelled by the assumption of 22 serial connected PV modules. Each module (Type TSM-PC05) has a peak performance of about 230 W, so the total peak performance of the



Fig 5: Photovoltaic system on the roof of the UdK main building with sensors for solar irradiation, wind speed, temperatures of air and modules

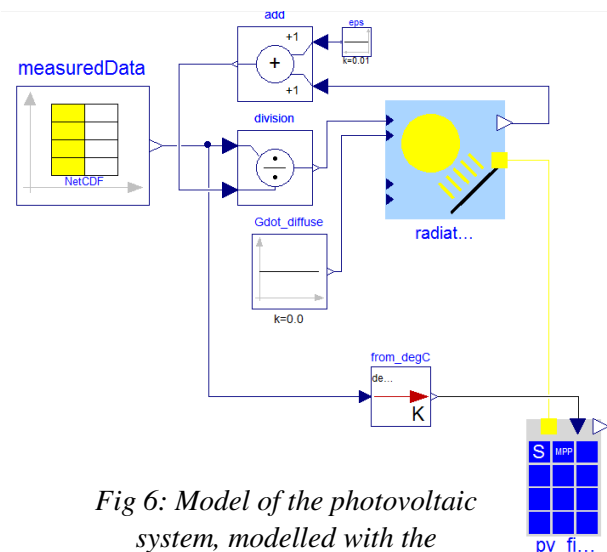


Fig 6: Model of the photovoltaic system, modelled with the *BuildingSystems* library

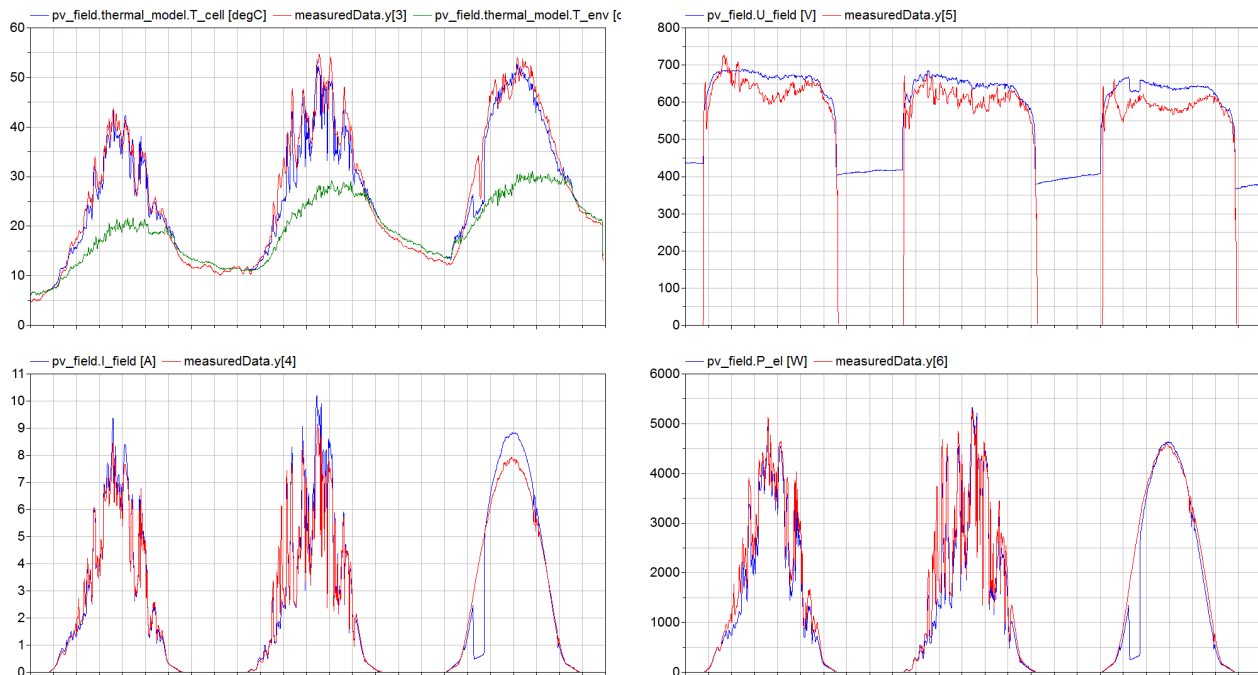


Fig 7: Comparison of measured and simulated values for three summer days:

- a) cell temperature: simulated (blue), measured (red); measured air temperature (green)
- b) string voltage: simulated (blue), measured (red)
- c) string current: simulated (blue), measured (red)
- d) string power: simulated (blue), measured (red)

string is about 5.060 W. The simplified model of a PV module of the *BuildingSystems* library was used, which works with an electrical one-diode model and an empirical thermal approach for the calculation of the cell temperature, depending on the air temperature of the cell environment [16].

Figure 7 shows the measured and the simulated values (temperatures, voltage, current and electric power) of the string of 22 PV modules during three summer days. All quantities have similar values for the real PV plant and for its simulation model. The cell temperature runs up two 20°C higher than the environment air temperature. Also the string voltage values are similar during the sunshine, in which the simulated values are higher than the measured values, the same goes for the current. After sunset the values of the simulation model are greater than zero, which is only a result of the modelling approach.

During the first two days the generated electrical power reaches the peak power for a short period. The collapse of the calculated electrical power during the morning hours is caused by a shading of the radiation sensor. The simulated performance drops

because of the measured radiation, but the real measured performance is not affected. The position of the sensor will be moved to a better place in the future.

It is typical for a one-diode model that the voltage and current values are higher than the real values, because a part of the internal losses of the PV cells is neglected. Therefore a constant correction factor is used in this model for the calculation of the power from the voltage and current. This factor is a model parameter that depends on the real qualities of a module (materials and construction). Unfortunately it can not be derived easily from the properties that are usually known.

With simulations using measured values of the real system this factor can be approximated. For the three days of the shown configuration a value of 0.82 proved to be ideal to bring the calculated electrical power close to the real (measured) values.

The plant is monitored permanently and all values are archived on a data server (see 4.1). This makes it possible to adjust the correction factor of the model with simulations using recent measurements. This task can even be done fully automated.

6 Conclusions and Outlook

The library was used for more than 10 years in a wide range of applications. It can be used with Modelica and other systems to access data in different ways. In combination with a DAP server it offers the remote access to different data sources.

The most used application today is reading weather data in Modelica simulations of buildings and solar systems. But it is easy to use the library for other purposes and with different software packages.

Although the library is in a stable state there are some possible improvements for the future:

- better integration with Modelica runtime systems (e.g. error handling and reporting),
- supplying information on derivatives of functions for improved integration performance,
- implementing optimizations for special cases like equally spaced grids,
- providing better tools for the conversion and preparation of data files,
- possibly including the library and its dependencies in Modelica systems (Dymola, OpenModelica, jModelica) to avoid the complex installation process on the different platforms by the user,
- finding a better name for the project. :-)

References

- [1] ncDataReader2:
<http://j-raedler.de/projects/ncDataReader2>
- [2] netCDF:
<http://www.unidata.ucar.edu/software/netcdf/>
- [3] DAP/OpenDAP:
<http://www.opendap.org/>
- [4] CSA:
<http://code.google.com/p/csa-c/>
- [5] Ernst, T., Klein-Robbenhaar, C., Nordwig, A., Schrag, T.: Modeling and simulation of hybrid systems with SMILE, in: Informatik, Forschung und Entwicklung, 15:33-55, 2000
- [6] ANSYS CFD:
<http://alturl.com/m8fjb>
- [7] LGPL:
<http://www.gnu.de/documents/lgpl-2.1.en.html>
- [8] Dmitrov:
<http://alturl.com/g8wzk>
- [9] METEONORM:
<http://www.meteonorm.com>
- [10] Pydap:
<http://pydap.org>
- [11] BuildingSystems:
<http://www.modelica-buildingsystems.de>
- [12] TRY:
<http://www.dwd.de/TRY>
- [13] Ljubijankić, M., Nytsch-Geusen, C., Rädler, J., Löffler, M.: Numerical coupling of Modelica and CFD for building energy supply systems, in: Proceedings of the 8th International Modelica Conference, 2011
- [14] DyMat:
<http://j-raedler.de/projects/DyMat>
- [15] Ljubijankić, M., Nytsch-Geusen, C., Jänicke, A., Schmidt, M.: Advanced analysis of coupled 1D / 3D simulation models by the use of a solar thermal system, in: Proceedings of the Building Simulation, 2011
- [16] Nytsch, C., Quaschnig, V., Scholtz, G.: Photovoltaik Modelle für die Simulationsumgebung SMILE, in: Tagungsband: 15. Symposium Photovoltaische Solarenergie in Staffelstein, OTTI-Technologiekolleg, Regensburg, 2000

Detailed geometrical information of aircraft fuel tanks incorporated into fuel system simulation models

Ingela Lind Alexandra Oprea

SAAB Aeronautics

SE 581 88 Linköping, Sweden

ingela.lind@saabgroup.com alexandra.oprea@saabgroup.com

Abstract

Fuel tanks in fighter aircraft have an irregular shape which is given by a detailed CAD model. To simulate a fuel system with sufficient amount of detail to solve the design issues, necessary geometrical information need to be given in a compact and computationally fast form. A function approximation using radial basis functions is suggested and compared with some other methods. The complete process from production scale CAD model to system simulation model is considered.

Keywords: aircraft design; fuel systems simulation; geometrical representation; surrogate model; radial basis functions

1 Introduction

A fighter aircraft fuel system is a system of many parts. Fuel fills up large parts of the aircraft not occupied with other equipment and the many different systems of the aircraft often pass through the tanks. To keep control of the center of gravity the tanks are divided into smaller parts and are interconnected by pipes. Fuel is pumped between the tanks to a collector tank which has a negative g-load compartment to enable the aircraft to fly inverted. The tanks are also pressurized to avoid evaporation of the fuel at high altitudes.

When designing aircraft fuel systems several issues demand detailed simulation models for analysis. The most important are

- Is it possible to provide the engine with fuel with enough pressure regardless of what pilot maneuvers and equipment faults that occur?
- Can the amount of accessible fuel be correctly determined at all times?

- Can the structural strength of the hull and all installation parts be estimated with good precision?

When these problems are solved, questions related to optimization of weight, fuel consumption, and heat storing capabilities as well as other issues need to be considered.

The fuel system simulation models needed to describe the system tend to be large (~400 state variables, ~16000 time-varying variables) due to the high number of parts involved. The combination of fuel (incompressible fluid) and pressurization air (compressible fluid) and the necessity to handle both fast time constants (as when a tank get full) and slow time constants (heat storage) make the models stiff and a bit awkward to work with. Still, the information gained from using the simulation models more than pays off the work spent to keep the models executable and is seen as a prerequisite for successful fuel system design work at Saab Aeronautics. A theoretical background on fuel system design can be found in [1] and how Dymola and other simulation tools are used in the system design process is described in [2] and [3].

In the past few years the idea of bringing geometrical information from CAD models into simulation models has gone from a distant dream to something actually achievable. To investigate if it was possible to get better accuracy of the fuel tank representation in the simulation models used, a study [4] was made to show the feasibility of extracting geometrical information from CAD models, do a function approximation of the data and then use the function in a tank model in Dymola. The work is not yet finalized to the level of inclusion in production processes, but the major steps and an evaluation of results are done. The intention is that the accuracy of the simulation models shall meet the measurement precision demands on the aircraft, and to improve the efficiency of the loads computations while simulation times are kept at the same level as before.

This can be done using two types of model improvements. First, the geometrical representation of the fuel tanks is changed from a simple two-dimensional square box to a full three dimensional representation of the complex geometry. Next, the aircraft model is changed from a point model where all accelerations act on the same point to a rigid body where accelerations in each tank depend on both the accelerations in the aircraft's center of gravity and the torque acting around it.

In this paper the major steps of the procedure will be discussed as well as later results showing that the process [4] can be scaled to full production size CATIA models as well as full size fuel simulation models.

2 Assumptions

A typical fighter plane fuel tank has a complicated shape; an example is shown in Figure 1. It is made up of many non-convex surfaces and even internal parts where bulky equipment is immersed in fuel. Due to the high order of complexity, describing the details of the fuel tank geometry in a simulation model is not feasible at the moment.

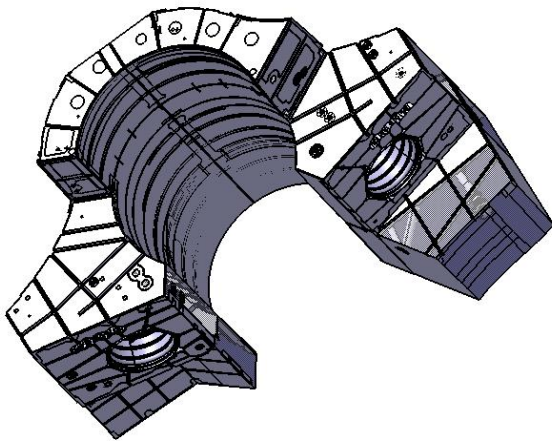


Figure 1 Typical body fuel tank shape, which is one of several different basic shapes. Note that equipment is immersed in fuel and fills up space within the tank.

It is assumed that the case is semi-static, such that the fuel surface is perpendicular to the acceleration vector of the tank at each time instant and that there is no fuel sloshing. Although a bit contradictory, it is also assumed that the fuel undergoes enough sloshing so that all fuel surfaces of the different compartments within the same tank are at exactly the same height, as if they were connected beneath the surface (which often is the case).

The information needed to perform a fuel system simulation is, given the orientation of the acceleration vector, the center of gravity of the fuel in a particular tank and the position of a point on the fuel surface. Both these vectors depend on the volumetric amount of fuel in the tank which is connected to the fuel mass state variable through the temperature dependent density. This means that it is enough to have a function that transforms the four variables fuelVolume and the 3D acceleration vector to the six variables given by the two 3D vectors representing the point on the fuel's surface and the fuel's center of gravity. The means to represent this function could be a table, but the assumption has been made that a function approximation would be both faster and less memory consuming.

3 Geometric data transformation

The first prerequisite for a fuel system simulation is the extraction of the geometric data from CATIA.

3.1 Extraction of geometric information

Each fuel tank in the Jas 39 Gripen has its own CAD model, and an analysis extracting the center of gravity (CG) and fuel surface data needs to be performed on each of them. The analysis itself consists of a macro written in VBScript. An early version of it can be found in [4]. In a nutshell, the steps of the analysis are as follows:

- Transforming the original fuel tank into a “dumb” solid, without construction history or identifiable individual features. This is done in order to reduce the file size, which affects the analysis time considerably.
- Creating the 2 bodies for the analysis: a “reference” body for comparison purposes and a “fuel” body on which the actual analysis will be performed. The dimensions of the “reference” body are recorded and saved.
- Creating the required elements for the analysis: the acceleration vector line, the sectioning plane perpendicular to the line which will split the “fuel” body in steps of pre-set length and the measurements on the “fuel” body which will update every time the body is sectioned.
- Performing the actual analysis. For each acceleration vector orientation in the aircraft's maneuverable range, the fuel body is split in steps of around 10 liters. For each split the

values of the resulting fuel volume, center of gravity and of the position of a point on the fuel's surface are saved to a text file.

3.2 Function approximation of data

The CATIA step generates around 20,000 to 40,000 distinct data entries for each fuel tank, with 10 parameters each. Although numerous, these are not enough since the simulation needs the CG and surface point coordinates for other acceleration vectors and volumes, as well. Therefore, a data interpolation step is required. To do the interpolation using Dymola interpolation tables would be feasible with a small amount of data in a low-dimensional case, but the high dimension and the amount of data calls for other methods. In this case the data interpolation is done using a parameterized function approximation called radial basis functions (RBF) networks. They are thought to be one of the best ways of approximation multi-variate scattered data, due to their excellent approximation properties [5], although in some cases vulnerable to the Runge phenomenon [8]. In short, they can be visualized as an "input - process - output" system. The input is the data generated from CATIA - the X , Y , Z orientations of the acceleration vector, the fuel volume, the X , Y , Z coordinates of the fuel surface and the corresponding ones of the tank's center of gravity. The output is a function, s , which can give a good approximation of the data for inputs different than the ones where the value of the exact function is known. The approximating function is defined using fewer points than the ones available in the input data (points which will be called centers). This means that a data reduction with maintained generalization ability is done. For visualization, see Figure 2.

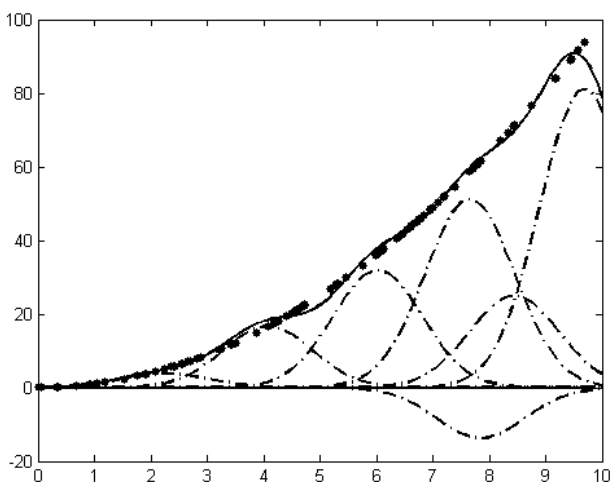


Figure 2 The Gaussian functions at the selected centers (dash-dotted) and the resulting approximating function (solid) based on the input data (points). In this

plot, the Runge phenomenon is visible at the right edge of the interval (the solid line drops), but it might also be a part of the explanation for the drop of the curve around $x=5$.

The middle layer, the so-called "process", is defined using the RBF functions themselves. The approximating function $s(x) = W_1 + \sum_{i=2...n} W_i f(k\|x_i - x\|)$ is

given by a weighted sum of a uni-variate function f with the Euclidian distance between the x_i and x as argument, where x are the points at which the approximation function is calculated, and x_i are the centers with respect to which the function is defined. The weighting factor W_i is associated to each center x_i . It is determined such that the error between the approximating function and the input data is minimized. The scaling factor k influences the support area of the function f .

The centers are selected using the orthogonal forward regression algorithm, presented in [6] and [7]. They could be selected at random, but the algorithm uses the modified Gram-Schmidt orthogonalization procedure to select the centers which minimize the error in the least-squares sense. The benefit of using the modified Gram-Schmidt method is that the resulting approximation is sparse in parameters. It starts from a large set of potential centers - basically, all the data resulting from the CATIA analysis, filtering them until the error sinks below a specified threshold. To be able to handle the very large data sets an approach where the data sets are divided into smaller pieces and the results recomputed several times is used, see [4]. When all data is processed, the algorithm returns the selected centers and their corresponding weights and writes them to a file to be used in the dynamic simulation in Dymola.

All RBF networks work according to the same principles. The differing factor is the function f , which makes up the linear combination defining the approximating function s . The ideal for the fuel tank problem, which is local and almost smooth in its nature, is to have a function with local support, so that new input data would not influence points situated far from it. Several possible functions have been tested to determine which ones are good choices with respect to sparseness of the parameters in the approximation and computational speed in the Dymola implementation. The typical number of parameters of the approximation is in the range 300 to 3000, a data reduction of a factor 100 to 1000.

A standard choice of a Gaussian function, see Figure 2, seems to fit the fuel tank problem best. There were concerns regarding computational effectiveness in Dymola as the exponential function is not considered cheap so other choices were considered.

Using $r = \|x_i - x\|$, the investigated options were

- the inverse quadratic function,

$$f(r) = \frac{1}{1 + (kr)^2}$$

- the inverse multiquadric function,

$$f(r) = \frac{1}{\sqrt{1 + (kr)^2}}$$

- the rational quadratic, $f(r) = 1 - \frac{r^2}{1 + r^2}$

- Wendland's functions,

$$f(r) = \begin{cases} (1-r)^5(5r+1), & r < 1 \text{ (or } 0.5) \\ 0, & \text{otherwise} \end{cases}$$

They all give less sparse results and need both more memory to store parameters and more multiplications to compute $s(x)$. Also B-splines outside a RBF framework have been considered, but do not fit ideally to high-dimensional non-latticed data.

3.3 Tank model implementation

Implementing the radial basis functions in the existing tank model of the commercial library Aircraft-FluidSystems (developed mainly by Modelon AB and partly by Saab Aeronautics) was a simple task. The only modification needed was the replacement of the existing distance computation between the position of the fuel surface and the tank ports of the connected pipes. This is now done using a scalar product and the approximating function $s(x)$.

The change in the acceleration vector definition brought by viewing the aircraft as a rigid body instead of a point mass was also straightforward.

4 Results

4.1 Influence on accuracy

A comparison between two fuel system simulations performed using the old two-dimensional tank representation and two simulations incorporating the new three-dimensional representation is presented in [4]. Both the 2D and 3D simulations are performed for two levels of accuracy of the CATIA analysis. The results show that there is only a minor difference in the system simulation precision between the two

CATIA target sampling accuracies of 18 and 12 liters. Compared to the CAD measurements of three different points for several acceleration vectors, the simulations resulted in an error of 3.3 kg for the 18 l precision (with a maximum error of 10 kg) and a 3.1 kg error for the 12 l precision (with a maximum error of 6 kg). For the two-dimensional tank representation the same average difference is 42 kg (with a maximum of 89 kg). It then follows that a three-dimensional representation of a moderate size makes a large improvement in the simulation accuracy of the fuel measurements. Comparison with a real fuel tank is not achievable since the tanks are not yet built, but it has been shown earlier that 'fuel measurements' in CAD models correspond well with fuel sensor calibration measurements in built tanks.

This accuracy improvement affects all parts of the fuel system simulation model, as the fuel flow through pumps and pipes depends on the fuel level in each tank.

4.2 Influence on simulation times

A comparison between the different possible kernel functions revealed that the initial choice of the Gaussian was the correct one. The evaluated functions, along with their training, computation times in MATLAB and simulation times in Dymola for a simple test model are given in Table 1. All the simulations are performed on the same tank, with the same parameters (the scaling factor $k=0.8$ where necessary and the error threshold set at 5 mm). The MATLAB computation time take only computation of the function $s(x)$ into account, while the Dymola simulation time also involves computations of all other equations necessary for a tank, two pipes and two sources in a test case.

Table 1 Comparison of training and simulation times for different kernel functions

Function	Training time, s	MATLAB computation time, s	Dymola simulation time, s
Gaussian	36.63	0.84	157
Inverse quadratic	58.85	1.12	190
Inverse multiquadric	139.26	4.15	161
Rational quadratic	63.57	1.11	155
Wendland, $r < 1.0$ mm	218.13	2.93	158
Wendland, $r < 0.5$ mm	318.08	2.95	176

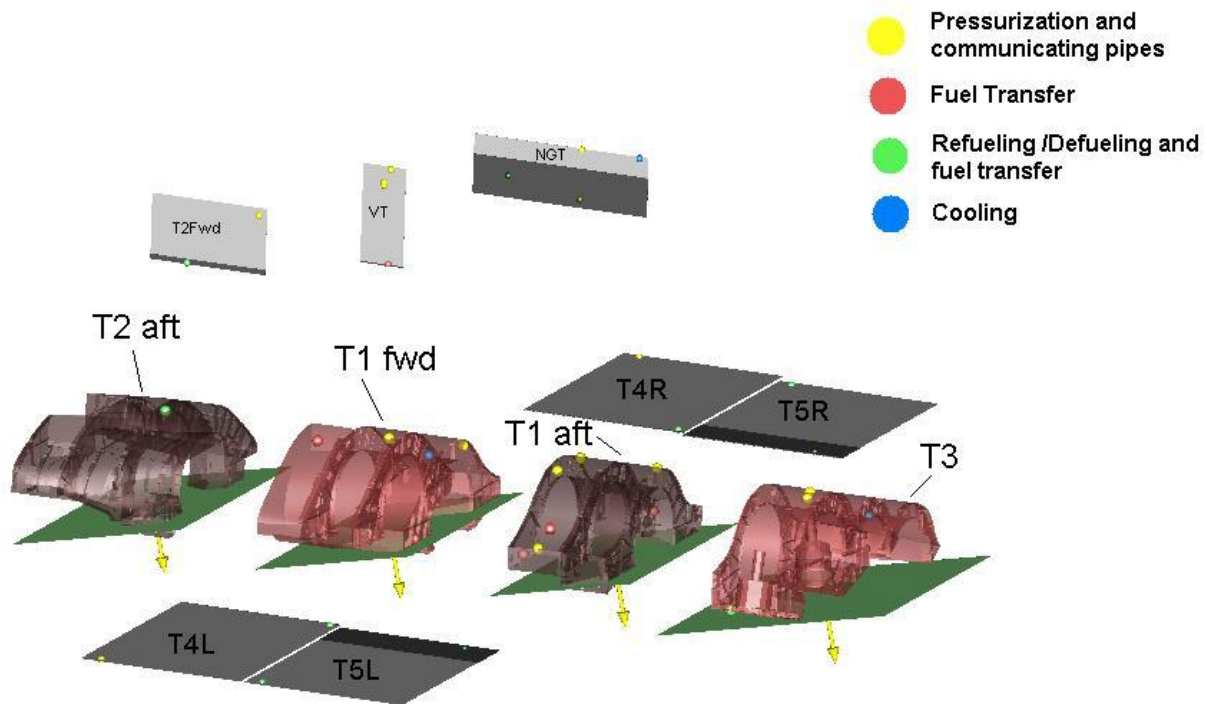


Figure 3 Visualization/animation of the fuel system using graphics from CATIA and simulation results from the fuel system simulation model. The green and black surfaces show the fuel level in the tanks. The yellow arrows show the acceleration vector for each tank and the colored balls show inlets and outlets of pipes for different purposes.

As for the variation of the Gaussian function with its defining parameters, simulations showed that there are optimum values of the scaling factor and of the error threshold. Any values different from these optimal ones lead to extended simulation times, without a significant improvement in accuracy. For the simulations with an error value of 0.01 mm, the MATLAB training time was between 3000 and 4000s, while for errors of 1 mm it dropped to several hundreds of seconds. The Dymola simulation results are summarized in Table 2. If the Runge phenomenon influences the function approximation, the simulation times easily grow a factor 10 or more, so care must be taken to avoid it.

Table 2 Dymola simulation times for different Gaussian settings

Settings	Dymola simulation time, s
k = 0.05, err= 0.01	166
k=0.4, err = 0.01	182
k = 0.8, err = 0.01	177

When a complete fuel system simulation model with four three-dimensional tank representations is compared to the same model having two-dimensional tank representations the times in Table 3 are achieved. The translation/compilation time depends

strongly on the number of defining parameters required by the function approximation, which is a reason to use sparse representations. The simulation time only grows by a factor 3, which is considered to be successful, given the higher accuracy of the results. The comparison case is representative of a typical simulation analysis.

Table 3 Comparison of simulation times

	Translation/compilation time (min)	Simulation time (min)
2D tank	1.5	10
3D tank	10.5	30

4.3 Influence on system insight

An animation of the tank models was implemented in order to identify what can be achieved, see Figure 4. This addition proved to be worthwhile from the very first simulations. The insight into the system behavior it provides shortens the time to learn the system. It is also a fast means of finding errors. For instance, one of the easiest errors to perform and most difficult to find is orienting the acceleration vector the wrong way. This can easily occur since

different departments use different coordinate systems and boundary conditions for simulations have many different sources. But in the animation this error is easily detected.

5 Conclusions

A full geometrical representation of fuel tanks at a given accuracy tailor made to accommodate fuel system simulation is no more a distant dream but a fully achievable task. The work has shown that

- it is possible to achieve an appropriate level of accuracy for all intended design studies,
- it is important to get a sparse representation (to keep the translation/compilation time down),
- several different choices of radial basis functions are usable and that the Gaussian is comparable to the others with respect to simulation time, but give more sparse representations,
- care is needed to avoid the Runge phenomenon [8] (which may slow down simulations considerably when the fuel level is close to a pipe end), and
- using RBF as function approximation keeps simulation times in the same level of magnitude as the simple and much less accurate 2D square box approximation previously used.

6 Acknowledgements

Jonas Wikström (Linköpings University) has dedicated his master's project to this project at Saab Aeronautics.

Sara Ekermann (Linköpings University) has worked with visualization and large-scale testing of results.

Thanks to Dassault Systèmes who has provided an evaluation license for the work in Catia as well as acted discussion partner of what is achievable.

References

- [1] Gavel. H. (2007) *On Aircraft Fuel Systems – Conceptual Design and Modeling*. Dissertation No.1067, Division of Machine Design, Department of Mechanical Engineering, Linköpings University. ISBN 978-91-85643-04-2
- [2] Lind. I. & Andersson. H. (2011) *Model Based Systems Engineering for Aircraft Systems – How does Modelica Based Tools Fit?* In proceedings of the 8th International Modelica Conference, Dresden, 2011
- [3] Steinkellner S., Andersson H., Gavel H. and Krus P. Modeling and simulation of Saab Gripen's vehicle systems, AIAA Modeling and Simulation Technologies Conference, Chicago, USA, AIAA 2009-6134, 2009
- [4] Wikström J., 3D Model of Fuel Tank for System Simulation: A methodology for combining CAD models with simulation tools, Masters thesis LIU-IEI-TEK-A—11/01201—SE, Linköpings University, 2011, <http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-71370>
- [5] Buhmann, M. D. Radial Basis Functions, *Acta Numerica* (2000) 1—38.
- [6] Chen. S., Billings. S.A. & Lou. W. (1989) *Orthogonal least squares methods and their application to non-linear system identification*. *Internal Journal of Control*, 50:5, 1873 -1896
- [7] Chen. S., Billings. S.A., Cowan. C.F.N. & Grant. P.M. (1990) *Practical identification of NARMAX models using radial basis functions*. *Internal Journal of Control*, 52:6, 1327 -1350
- [8] Boyd, J.P., *Six strategies for defeating the Runge Phenomenon in Gaussian radial basis functions on a finite interval*. *Computers and Mathematics with Applications*, 60 (2010), 3108-3122.

Simulation of Artificial Intelligence Agents using Modelica and the DLR Visualization Library

Alexander Schaub Matthias Hellerer Tim Bodenmüller
German Aerospace Center, Robotics and Mechatronics Center
Münchner Straße 20, 82234 Weßling

Abstract

This paper introduces a scheme for testing artificial intelligence algorithms of autonomous systems using Modelica and the DLR Visualization Library. The simulation concept follows the 'Software-in-the-loop' principle, whereas no adaptations are made to the tested algorithms. The environment is replaced by an artificial world and the rest of the autonomous system is modeled in Modelica. The scheme is introduced and explained by using the example of the ROboMObil, which is a robotic electric vehicle developed by the DLR's Robotics and Mechatronics Center.

Keywords: Simulation of Artificial Intelligence Agents; Autonomous Systems; Software-in-the-Loop; DLR Visualization Library; ROboMObil

1 Introduction

The variety of autonomous systems, or also known as artificial intelligence agents (AIA), can range from small toys like Lego mindstorms to full-sized robotic cars like the ROboMObil (ROMO)[1]. In all cases an agent consists of three essential parts: sensors, the core artificial intelligence for the agent's functionality, and actuators [2]. The agent perceives its current environment through its sensors, interprets it and plans the next actions to reach its goal before acting upon the environment through its actuators. For a sufficient simulation of an autonomous system the bidirectional connection of an agent to its environment must be considered.

In the past decade several open source simulation environments for autonomous systems, mostly for robots, have been launched due to increasing computational power and decreasing hardware costs, which have made the use of autonomous (mobile) robots feasible for education.

Published in 2001, the socket-based device server Player in combination with the multi-robot systems

simulator Stage [3] was widely used in academia and industry. Player provides simple TCP sockets to external devices like sensors and actuators. Player is language neutral and uses the UNIX abstraction of devices being considered as files. Stage is a simulation environment for multiple robots with computationally cheap, but in terms of fidelity only sufficient models. The linear scaling with the population of the simulated world was very important. It is a 2D simulator for indoor scenarios. The simulated sensors are rather simple laser range finders or sonar than complex sensors like cameras.

In 2003 Gazebo [4] was released to satisfy the need for a 3D simulation environment for Player. It enables the simulation of cameras, uses rigid body models, and still works, despite the increased complexity, with simulating several autonomous systems concurrently.

Nowadays the Robot Operating System ROS [5] is the most popular environment for connecting algorithms, sensors, and actuators of robot systems. Many functions and drivers were adopted from Player. Moreover, it also uses interfaces to Stage for 2D and to Gazebo for 3D simulations.

Microsoft's Robotics Developer Studio [6] is a free, but not open source, development suite using user friendly techniques for visual programming, easy parallelization, and debugging via web-interfaces. It is equipped with a DirectX based Visualization, its own rigid-body physics engine, and provides interfaces to commercial products from FischerTechnik, iRobot, Lego etc.

Furthermore, there are also several commercial robot simulators like the Virtual Robot Experimentation Platform V-Rep [7] or Webots [8].

Proprietary simulation environments were developed for larger projects like Junior - Stanford's robotic car for the DARPA Urban Challenge [9]. That proprietary software can be adapted to special demands, which are not completely fulfilled by generalized tools like the ones named before.

All of the mentioned simulation environments use physics engines like Bullet Physics Engine [10] or Open Dynamics Engine [11], which have a strong gaming or computer animation background and provide rigid body modeling and collision detection. They try to reach a fast computation while providing a sufficient accuracy of the physics. Modelica provides several advantages being able to model complex physical systems containing e.g. flexible-bodies, electrical and hydraulic components. To be used for the simulation of artificial intelligence agents Modelica has to be extended by an advanced visualization like the DLR Visualization Library [12]. The combination of Modelica with the DLR Visualization Library creates a powerful tool for an efficient development of complex physical agent and environment models.

Our motivation for the presented scheme is a bidirectional autonomous systems simulation, which combines complex Modelica models of the ROMO with the artificial intelligence system used in the real vehicle.

The remaining chapters are organized as followed: The second chapter provides an overview of our simulation concept. Chapter three gives a detailed explanation of the used tools and interfaces. Afterwards, the results of a simple example will demonstrate the functionality of the AIA simulation scheme. Finally, we will conclude with a brief summary and outlook.

2 Concept of the AIA Simulation

The main target for the proposed scheme is a 'Software-in-the-Loop' simulation, which means that no changes are made to the algorithm that should be tested. In order to test the artificial intelligence of an autonomous system the perception, planning, and control algorithms are kept and its hardware and the environment are simulated. The system's hardware is substituted by a Modelica model, where the detail of the model varies depending on the purpose of the simulation. It can range from a rigid body model to an overall system model containing electrical, flexible, hydraulic, thermal, and tire (sub-)models.

The second step is the replacement of the environment by using the DLR Visualization Library, which extends Modelica by an advanced visualization and interactive simulation. Standard sensors for velocity, torque etc. are part of the basic Modelica library, but complex perception sensors like cameras require this advanced visualization for a sufficient simulation. The algorithms tested with this scheme and also their inter-

faces to the rest of the autonomous system do not have to be changed. Hence, the algorithms have to run outside the Modelica environment during the simulation, which is made possible by the interactive interfaces provided by the DLR Visualization Library.

The proposed simulation scheme using the example of the ROMO is depicted in Figure 1. The main distinction is made between the autonomy hardware and the simulation hardware. Both can run on the same PC, but the hardware of the autonomous system usually consists of several connected processing units. The intention is to follow the 'Hardware-in-the-Loop' principle and to connect the AIA system to a simulation PC.

The primary perception sensors of the ROMO are cameras, which are widely used in modern autonomous systems, as they provide a great variety of information [13]. A typical cycle of the scheme starts with the virtual cameras taking images of the simulated environment. The images and other sensor data is packed according to the SensorNet format and passed into the shared memory of the autonomy hardware. The interface from the Visualization library to SensorNet is described later in detail. Different algorithms that process and interpret those data can access the shared memory concurrently. The processed data is passed to the planning module both directly and via a module that updates the environment representation. The planned trajectory and other control data is passed via an interactive interface to the Modelica model. Sensors are triggered and the controller gets its reference input. In this example the vehicle dynamics controller is nested in the simulation module, since it does not run on the same hardware as the autonomous driving components in the real vehicle. With the controller commanding the actuators the ROMO model moves in the virtual world and the loop is closed.

Such a 'Software-in-the-loop' scheme for autonomous systems has several advantages. It is possible to test an algorithm under reproducible settings, which is usually not the case in reality. The camera-based perception is very sensitive to changing light conditions. Additionally, it is difficult to keep relative positions and velocities of objects the same in every test. The reproducibility is also desirable for comparing different algorithms and an essential requirement for reverse engineering.

Moreover, algorithms can be tested with optimal conditions. At the very beginning of an algorithm development it is helpful to see if the general concept is working while neglecting sensor and actuator noise

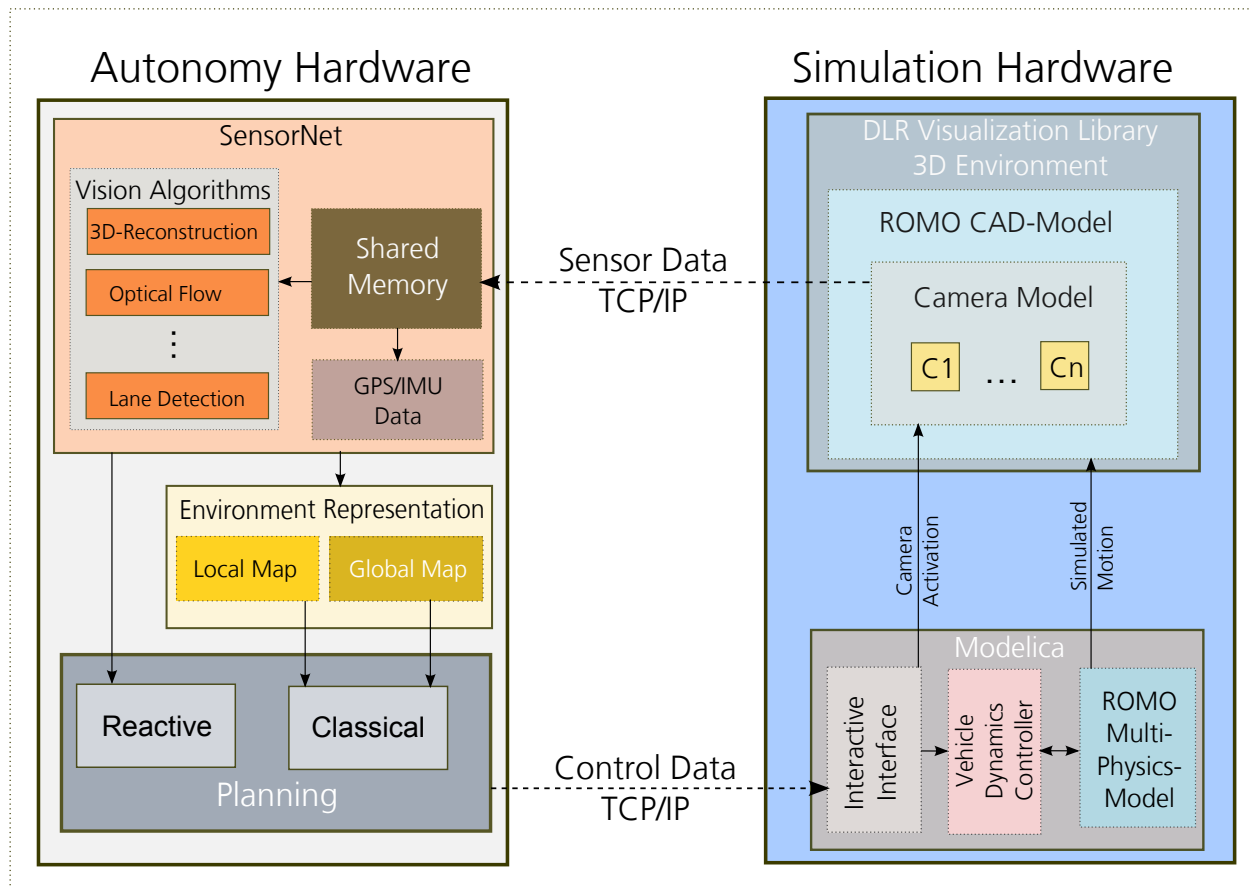


Figure 1: ROMO AI-Simulation Concept

and other disturbing influences. After the basic functionality has been proven the noise can be increased step by step to test different levels of robustness.

Another advantage is the adjustable level of abstraction. Autonomous driving software can be evaluated firstly with a simple model of the system's dynamics. Limitations of actuators can be neglected, sensors can be considered as all knowing and physical constraints can be softened. During the development process the model complexity can be raised to achieve a more realistic behavior of the simulated system. Furthermore, the developer can build a virtual world according to his needs. New types of sensors and systems can be modeled that do not exist yet. The preparation and execution of tests done by the simulation scheme is much faster than tests in reality. The simulation can be run faster than real time causing less costs and posing no harm for people and equipment. Nevertheless, there is still the need for tests with the real system, but their frequency can be considerably decreased. Hence, the 'Software-in-the-Loop' principle is very helpful for rapid prototyping.

3 Combining Virtual Reality with Perception

An essential part of the proposed simulation concept is the link between 3D simulation provided by the DLR Visualization Library and image processing algorithms, which utilize SensorNet for image data dispatching.

3.1 The DLR Visualization Library

The DLR Visualization Library is an extension to Modelica for 3D visualization of simulations. It is composed of two parts: a Modelica library and a standalone program.

The library part defines Modelica multi-body elements which do not influence the simulation's physics but are used for configuration of the simulation's visualization. The visualization is then displayed in a separate application called SimVis. An example of this can be seen in Figure 2. On the left it shows a Modelica model using the DLR Visualization Library library and on the right the corresponding visualization in SimVis. The DLR Visualization Library library provides a

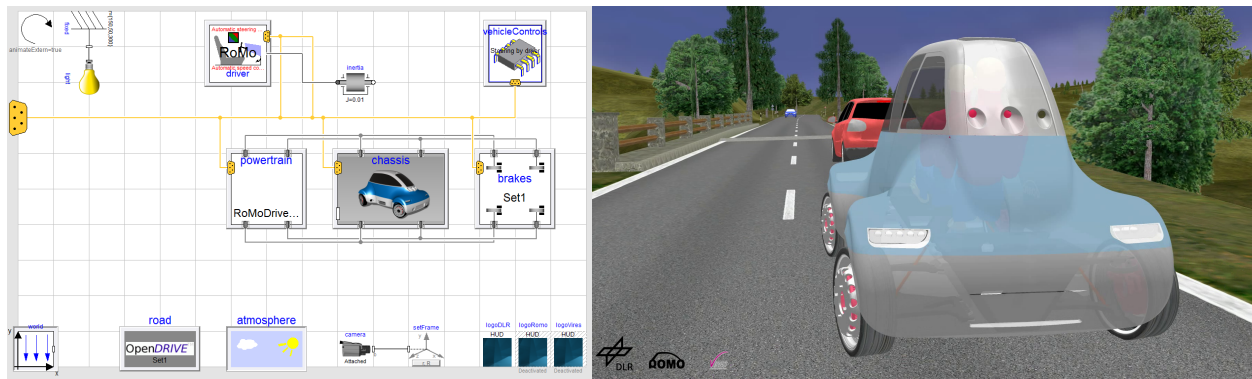


Figure 2: A Modelica model of the ROMO and the corresponding visualization

wide range of 3D objects from simple elements like boxes and gearwheels to complex 3D files to objects defining the representation like Head-Up-Displays showing variables or camera positions both in the 3D environment and their images on the screen.

From a technical perspective this is achieved by utilizing Modelica's C language interface to establish a TCP/IP connection between the simulation and the SimVis application, transmitting information about the configuration of the 3D elements to be displayed [12].

3.2 SensorNet

Modern robotics applications often use cameras and require the real-time analysis of images. The problem for this application is twofold:

First the amount of data is immense. For example a single VGA camera generates about $640 \cdot 480 \cdot 3\text{Byte} \cdot 30\text{Hz} = 28\text{MByte}/s$ of raw data. Moreover, recent video compression methods, e.g. mpeg4 or divx, are computationally expensive and also degenerates the image quality and therefore should be avoided in image processing tasks. Additionally, robots interact with their environment. Therefore, real-time restrictions apply to the image processing. The time from image acquisition to a possible reaction has to be minimized. This requires extremely efficient dispatching of image data which is achieved by the communication framework SensorNet. It is designed to provide sensor data, e.g. from cameras, with low latency to multiple, concurrent applications. Therefore, previous concepts on local real-time communication via shared memory [14] and on unified description of camera and range sensor data [15] are combined and extended in the SensorNet data streaming concept. In detail, a ring buffer on a shared memory in conjunction with a signaling mechanism is used to distribute data from a server process to multiple client processes with low

latency ($<100 \mu s$). The interface also comprises data type metadata that allows for type checking. Further, predefined, unified data types are used, e.g. color image or depth image, and act as an abstraction layer. As a result, sensors of same type can easily be exchanged by just replacing the server process. Data can be distributed across system borders by connecting shared memories on the different systems with UDP- or TCP-based data transfer. Additionally, a separate TCP-based configuration channel allows for setting and getting parameters, e.g. camera shutter time, without influencing real-time data streaming.

3.3 Interface

Acquiring images with real cameras under controlled conditions is not always feasible as described above. The intention is to reuse the existing solutions for 3D simulation and image data dispatching.

The DLR Visualization Library cameras are designed for displaying images on screen. Since Modelica is an object-oriented language the new camera model is derived from the existing solution and extended by additional parameters. The cameras are by default aligned within the 3D environment using rigid body transformations and displayed either in the SimVis window or full screen. In both cases the camera resolution is determined as a ratio of either the window size or the screen size for easier portability from one PC to another. In contrast real cameras have a fixed resolution in pixels. The simulation therefore requires this resolution as new parameter. Likewise the image data is always displayed on screen in RGB format, yet cameras often use different formats. This implementation currently supports two alternative formats: YUV and grayscale. Furthermore, SensorNet needs a name for the shared memory object to identify a specific camera and a role for the camera in a stereo setup. If the cam-

era is part of a stereo setup, both cameras use the same shared memory object. One camera has to be set as master and one camera has to be the slave. Both images will then be acquired simultaneously and put in the same shared memory object, whenever the master camera is triggered.

With these additional parameters set up in the DLR Visualization Library, SimVis can also reuse the majority of its existing camera implementation. The main difference lies in the render target. Normal cameras render to a frame buffer that is then displayed on screen. For the simulated cameras the render target is redirected to a frame buffer object (FBO), which is not displayed but read back to the main memory. Thereby, the image is rendered the same way but off-screen and accessible by the application as a data array in RGB format. This image data first has to be converted to the desired image format. Conversion into YUV is carried out using the following equation per pixel:

$$Y = 0.299R + 0.587G + 0.114B$$

$$U = 0.493(B - Y)$$

$$V = 0.877(R - Y)$$

This equation is also applicable for conversion in grayscale by only using the Y component describing the pixels luminance [16]. The preprocessed image is then packed into one of SensorNets default image formats, a time-stamp corresponding to the current simulation time is added and the image is released. Releasing an image in the SensorNet context makes it available for other applications. The primary focus lies on image interpretation algorithms that are part of an artificial intelligence agent.

4 Experimental Results

The proposed scheme is evaluated by a simple example, in which a testing environment for a vision based control (VBC) platooning algorithm is created.

The basic idea is that the ROMO follows a preceding car, while using only a front stereo camera pair for perception. Initially, the ROMO has only an image of the back of the target car, which was taken from an appropriate distance for following.

After the platooning mode is activated the ROMO tries to find the target car in the current camera image. Analog to vision based robot control [17] the goal is to see the target object in the same size and at the same angle as in the reference image. Therefore, the ROMO tries to reach and hold the very same relative position

in which the initial image was taken. The target position can also be made velocity dependent for keeping the minimum safety clearance.

A simulation model in Dymola is created. First, the ROMO's top front camera pair, refer to Figure 3, is modeled by using the DLR Visualization Library's SensorNet camera class, which was developed for the proposed simulation scheme, with the appropriate parameterization. They are attached at their respective positions to a 3D geometry model of the ROMO, which is extracted from CAD. An overall model of the ROMO containing all actuators, sensors, the electrical systems etc. can be used, but the example focuses on the perception part, which is the main interest in this paper. Buildings, streets, and a surrounding land-



Figure 3: The DLR's ROboMObil

scape are placed in the virtual environment. For this the DLR Visualization Library provides an integration block for common 3D files like the '.3ds' format. The target car consists of an animated 3D model bound to a trajectory block that moves the object within the virtual world. Now the simulation is started and images are sent to the shared memory via SensorNet. The AIA algorithms receive a notification that new images are available and begin to run. First, a SensorNet implementation of the DLR's 3D reconstruction algorithm, called Semi-Global Matching (SGM)[18], calculates depth information out of the two images from the stereo camera. The result can be seen in Figure 4, whereas the left part shows the scene recorded by the stereo cam and the right part a visualization of the depth values. The color value of every pixel is set ac-

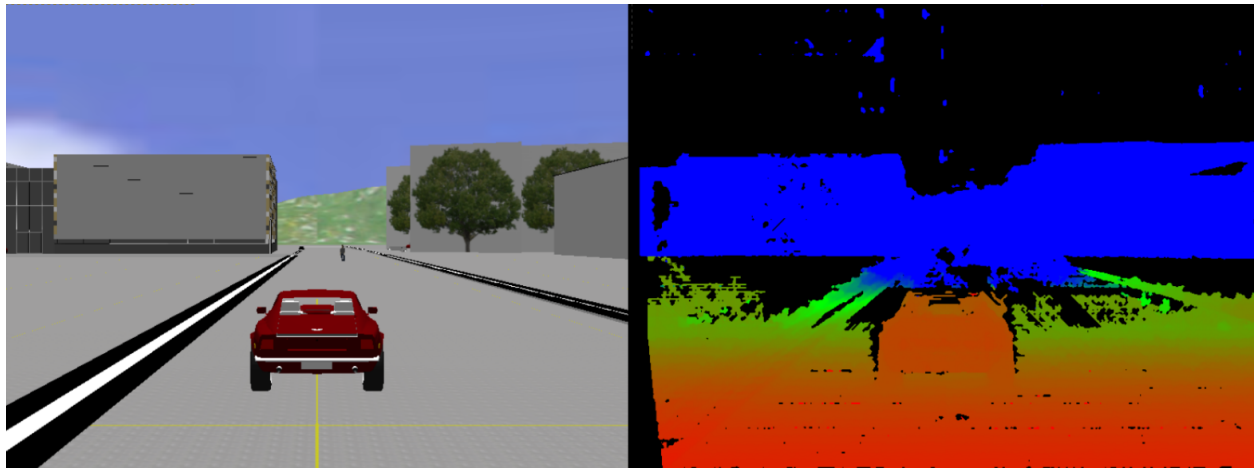


Figure 4: Semi Global Matching applied to the virtual images

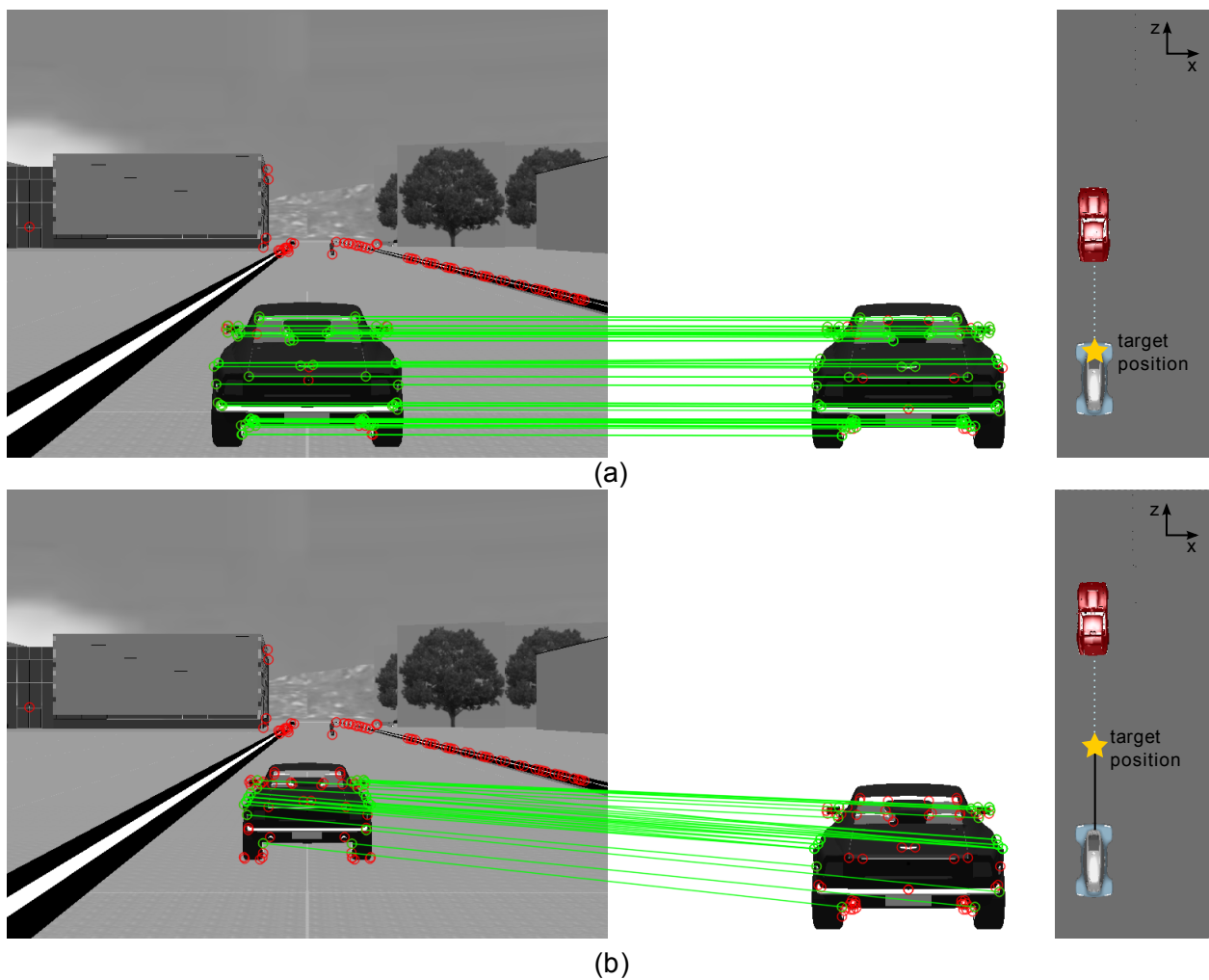


Figure 5: Matching features for estimating the relative position: (a) At target position (b) 4 meters deviation in camera direction

According to the depth value at that point. Small values are colored red and with increasing depth they go from orange to green to blue. Black parts of the depth image are either too far away like the sky or cannot be recon-

structed. This is often the case in regions with homogeneously textured surfaces, where the reconstruction algorithm cannot find matching points in both images. After calculating the depths the SGM writes a struc-

ture consisting of a rectified actual image, a quality map, and the depth image back into the shared memory. This structure is accessed by the VBC car following algorithm, which starts with running a feature detection algorithm, e.g. AGAST [19], on the actual image. A descriptor for matching is calculated for every feature point and the 3D coordinates of every feature point are determined using the depth image from SGM. The keypoints of the target image with their descriptors and 3D values are available a priori and so descriptors are compared to find matches in both images. The results can be seen in Figure 5, whereas the right is the target and the left one is the current camera image of the simulation. Matching keypoints are connected with a green line. Besides the markings there is no color in Figure 5, as the feature detection works with grayscale images. At least four matching keypoints are randomly selected. By using their respective 3D coordinates the rotation matrix R and translation vector T between the current keypoints and the target keypoints are calculated. The quality of the estimated R, T is measured by applying R, T to all keypoints of the current image that have matches. After that they are projected back into the 2D image space and the distance to their matching points in the target image is measured and summed up over all keypoints. The whole procedure is repeated with other sets of features until the quality of R, T is sufficient or a certain number of iterations is exceeded and the best iteration will be kept.

The preceding car in the simulation starts at the target relative position and moves four meters in the camera's z direction, whereas the ROMO remains stationary. The matches at the beginning can be seen in Figure 5a and that the end of the movement is depicted in Figure 5b. The number of found correspondences decreases during the movement. This is normal on the one hand due to the changed perspective, but on the other hand it is additionally disadvantaged here by the simple textures, which lead to weak feature points. Incorrectly matched or too few feature points can disturb the results of the algorithm immensely.

Nevertheless, the simulation has shown the general functionality of the algorithm as can be seen in Figure 6. The z value of the deviation to the goal position changes from 0 to 4000mm, while the target car moves in z -direction. The deviation to the real position can be due to badly chosen feature points, depth measurement errors, or imprecise calibration of the virtual cameras. Based on the computed rotation and translation a trajectory can be calculated and fed back into

the Modelica simulation via a TCP/IP channel to control the simulated ROMO in its virtual environment. In this early state of the algorithm's development the AIA simulation scheme is very helpful to identify weaknesses and increase robustness before tests with the real ROMO are possible.

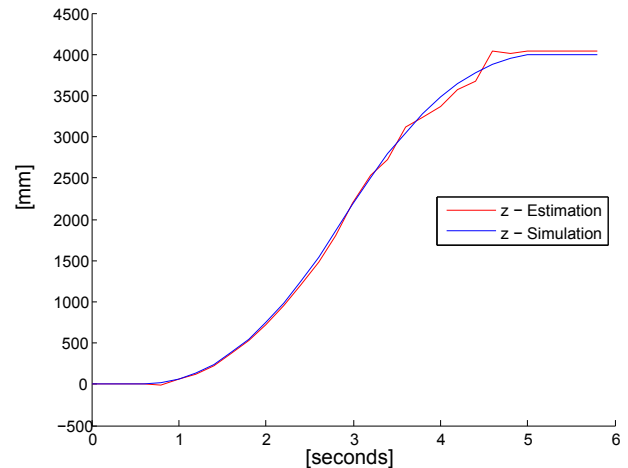


Figure 6: The deviation to the relative target position. z is in camera direction

5 Conclusions and Future Work

This paper presents a scheme for testing artificial intelligence algorithms for autonomous systems according to 'Software-in-the-loop' and 'Hardware-in-the-loop' principles. Existing multi-physics models are combined with the actual artificial intelligence algorithm that do not have to be adapted for the simulation. This is achieved by extending the DLR Visualization Library by an interface to the sensor data management tool SensorNet, which is utilized in real autonomous systems in DLR's Robotics and Mechatronics Center. The capability of the concept is proven by a short example, in which the translation and rotation to a leading vehicle are determined by a vision based car following algorithm.

In further developments more sensor types, which are typically used in autonomous systems like a dGPS aided Inertial Measurement Unit (IMU), will be modeled. Camera models can be extended with more realistic effects such as lens distortions. Moreover, we plan to use virtual objects with more complex textures to generate more realistic virtual pictures. In order to validate the simulation results they have to be compared to those using data taken from vehicle tests.

Acknowledgements We thank the following colleagues for supporting us with their expert knowledge: Darius Burschka (image processing), Martin Otter (Modelica), Tobias Bellmann (DLR Visualization Library), Heiko Hirschmüller (SGM), Klaus Strobl (camera calibration), and the whole ROboMObil Team.

References

- [1] Jonathan Brembeck, Lok Man Ho, Alexander Schaub, Clemens Satzger, and Gerhard Hirzinger. Romo - the robotic electric vehicle. In *22nd International Symposium on Dynamics of Vehicle on Roads and Tracks*. IAVSD, 2011.
- [2] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, third edition, December 2009.
- [3] B. Gerkey, R. Vaughan, and A. Howard. The player/stage project: Tools for multi-robot and distributed sensor systems. In *11th International Conference on Advanced Robotics (ICAR 2003)*, Coimbra, Portugal, June 2003.
- [4] N. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *Intelligent Robots and Systems, 2004. (IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 3, pages 2149 – 2154 vol.3, sept.-2 oct. 2004.
- [5] Morgan Quigley, Ken Conley, Brian P. Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.
- [6] J. Jackson. Microsoft robotics studio: A technical introduction. *Robotics Automation Magazine, IEEE*, 14(4):82 –87, dec. 2007.
- [7] Marc Freese, Surya P. N. Singh, Fumio Ozaki, and Nobuto Matsuhira. Virtual robot experimentation platform v-rep: A versatile 3d robot simulator. In Noriaki Ando, Stephen Balakirsky, Thomas Hemker, Monica Reggiani, and Oskar von Stryk, editors, *SIMPAR*, volume 6472 of *Lecture Notes in Computer Science*, pages 51–62. Springer, 2010.
- [8] O. Michel. Webots: Professional mobile robot simulation. *Journal of Advanced Robotics Systems*, 1(1):39–42, 2004.
- [9] M. Montemerlo, S. Thrun, and et al. Junior: The stanford entry in the urban challenge. *Journal of Field Robotics*, 2008.
- [10] <http://bulletphysics.org> - 15.05.2012.
- [11] Russell Smith. Open dynamics engine, 2008. <http://www.ode.org/>.
- [12] Tobias Bellmann. Interactive simulations and advanced visualization with modelica. In *Proceedings 7th Modelica Conference, Como, Italy*, 2009.
- [13] Alexander Schaub, Jonathan Brembeck, Darius Burschka, and Gerd Hirzinger. Robotic electric vehicle with camera-based autonomy approach. *ATZelektronik*, 2(2):10–16, April 2011.
- [14] G. Hirzinger and B. Bauml. Agile robot development (ard): A pragmatic approach to robotic software. pages 3741 –3748, oct. 2006.
- [15] T. Bodenmuller, W. Sepp, M. Suppa, and G. Hirzinger. Tackling multi-sensory 3d data acquisition and fusion. In *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pages 2180 –2185, 29 2007-nov. 2 2007.
- [16] Charles Poynton. *Digital Video and HDTV Algorithms and Interfaces*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1 edition, 2003.
- [17] F. Chaumette and S. Hutchinson. Visual servo control. i. basic approaches. *Robotics Automation Magazine, IEEE*, 13(4):82 –90, 2006.
- [18] H. Hirschmuller. Stereo processing by semiglobal matching and mutual information. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 30(2):328 –341, 2008.
- [19] Elmar Mair, Gregory D. Hager, Darius Burschka, Michael Suppa, and Gerhard Hirzinger. Adaptive and generic corner detection based on the accelerated segment test. In *Proceedings of the 11th European conference on Computer vision: Part II, ECCV'10*, pages 183–196, Berlin, Heidelberg, 2010. Springer-Verlag.

Functional Development with Modelica: A Use-Case Analysis

Stefan-Alexander Schneider* Tobias Hofmann†
stefan-alexander.schneider@bmw.de hofmann.tobias@me.com

* BMW AG, 80788 München, Germany

† TU München, Germany

Abstract

This contribution deals about the development steps of an embedded controller. The activities of the role function developer are explained for the simple example traffic light controller. The method of virtual integration is explained to establish short feedback loops.

Keywords: embedded systems; simulation; modeling; short feedback loops; co-simulation; virtual integration; Vee-Model; systems engineering

1 Introduction

The behavior of a dynamic system is in general too complex to treat by theory or formulas. Several simulation methods have been established for analyzing such systems. The virtual integration method is conducted on a model to gain knowledge about the (intended) real system behavior. This abstraction typically allows to focus on the main properties of the studied multi-domain system and their effects. These components require specific domain solvers for mechanical, electrical, etc. components. In this context, the term co-simulation has been established. The virtual integration is based on co-simulation and described in [7, 10]. There is a rather huge literature on the Vee-Model and systems engineering, see e.g. [1, 6, 12, 9, 4]. For more general introduction see, e.g., [5, 15, 16].

In the following, we demonstrate how to develop a control algorithm for an embedded controller designing the entire system - both the plant and the control components - with the modeling language Modelica. This approach allows us the modeling and simulation of the entire system, and thus the validation of the design decisions in an early phase of the development.

2 Model Example

Traffic is in general a good example for dynamic systems. The planning of traffic flow includes among others the avoiding of traffic jams and the optimization of traffic flows. No wonder that traffic planning is a current political issue as the article *Guck mal, wer da fährt* in the *Süddeutsche Zeitung* of May 15th 2012 shows. According to this article, the traffic of a city like Munich is controlled by more than 1.000 traffic lights. All these traffic lights serve to control the traffic and arrange for all traffic participants in some sense optimal traffic flow and an acceptable (system) behaviour.

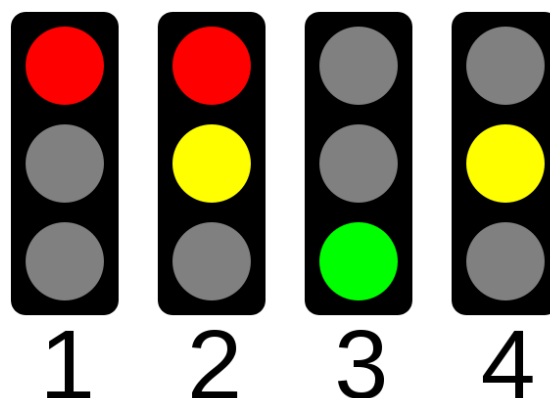


Figure 1: The typical sequence of coloured lights, see table 1.

Figure 1 explains the typical European sequence of coloured lights, see, e.g., [14].

traffic light	meaning
red light	do not cross
red and yellow light	prepare to cross
green light	cross
yellow light	if safe to do so, stop

Table 1: The typical sequence of coloured lights and their meanings.

A signal timing plan is a graphical representation of the traffic light phases for the corresponding traffic lights, similar to a so-called GANTT chart, see also Figure 2.

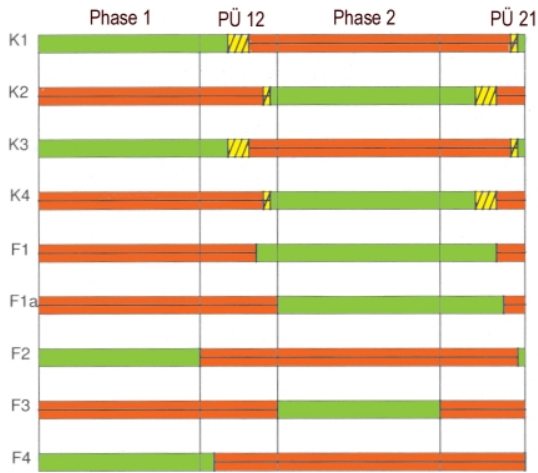


Figure 2: A typical signal timing plan is a graphical representation of the traffic light phases similar to a GANTT chart.

Traffic engineering programs like LISA+, see Figure 3, facilitate a planning process and are especially developed for intersections with a large number of signal groups and traffic lights, see, e.g., see [13].

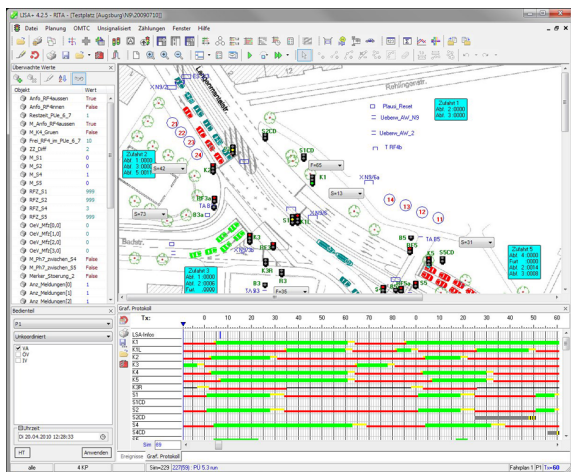


Figure 3: The GUI of the software package LISA+ for the planning of a specific traffic scenario.

Although the analysis of such systems of traffic lights contains a number of interesting (non-linear) mathematical tasks, we simplify the considered task to a single two crossing road intersection. The main reason for this is that we can better study the phases of the development process for such a simple example.

In this report we therefore restrict to the following model example: a simple intersection of two roads with four traffic lights, see Figure 4. According to the wind rose, the lanes are denoted by *North*, *East*, *South*, and *West*.

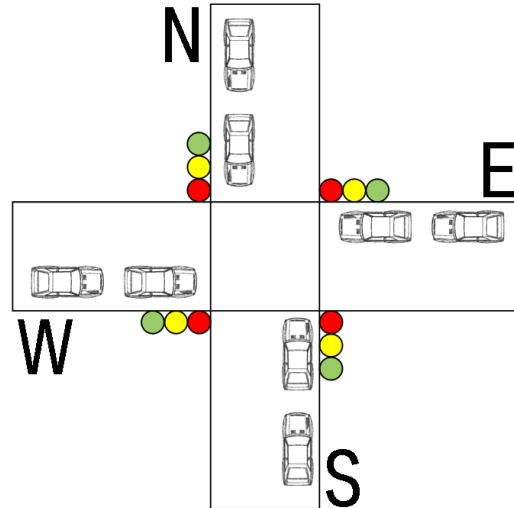


Figure 4: A simple road junction serves as for this paper sufficiently model example where the road crosses a north-south direction with a road in east-west direction.

We describe and study in the following sections a workflow with its development phases for an embedded control system for the traffic lights, using among other the environment design, modeling and simulation language Modelica and its modeling and simulation tool Dymola.

3 The Development Phases

The development phases of the Vee-model that are considered in this paper are, see Figure 4, [17]: system level requirements, system design, module design, module implementation, module integration and test and finally system integration and test on an embedded controller.

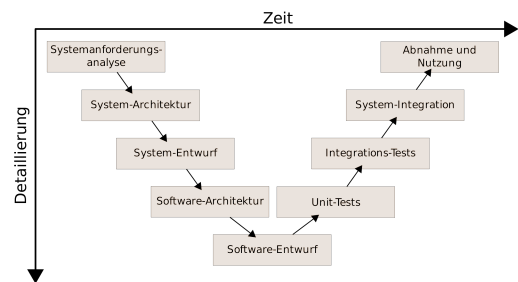


Figure 5: The Vee-model and its development phases.

3.1 System Level Requirements

In this development phase we formulate the requirements to the system and with that to the controller to be developed.

For this purpose, we define the *waiting time* W_i of a single vehicle crossing as the time from arrival at the intersection to the crossing of the intersection and with that leaving the system. We define the *total waiting time* by the overall sum $W = \sum W_i$ and formulate with that the first requirement to the intended control algorithm:

$$\Delta W := W_{new} - W_{old} \rightarrow \min, \quad (1)$$

where W_{new} denotes the overall waiting time of all vehicles after and W_{old} before the considered cycle of green phases. Although W_{new} and W_{old} are hard to measure, the difference ΔW , however, is not: the difference is depending on the number of vehicles crossing the intersection in the considered green phase cycle. This first requirement has the consequence that control algorithms with so-called vacant green phases are rated worse.

Let us now assume a scenario with high traffic rate. In this case, there exists a simple strategy to avoid vacant green phases by just not switching the priority lane. A controller that serves only one direction has no vacant green time and therefore fulfills the first requirement.

We formulate therefore a second requirement to prevent this undesirable behavior:

$$\text{Both directions are to be served periodically.} \quad (2)$$

We denote the *green times* $t_{NorthSouth}$ and $t_{EastWest}$ for the two directions, the *minimum green time* by t_{min} and the *circulation time* t_{Clock} by the sum of all traffic lights phases. Therefore holds

$$0 < t_{min} \leq t_{NorthSouth}, t_{EastWest} < t_{Clock} \quad (3)$$

and with that $2 \cdot t_{min} \leq t_{Clock}$.

3.2 Outlook: Additional Requirements from Functional Safety

Finally, note that there are additional requirements e.g. from functional safety:

1. *emergency control mode*: Traffic lights from the major roads turn off and the traffic lights from the side streets blink yellow. This indicates that the proper operation of the traffic lights is not guaranteed and supports on the other hand the given traffic signs.

2. *secure on the electrical level*: If a light source is out of order, so none of the directions may be given the green signal to avoid a so-called *hostile green* and it should, if possible, the red signal be given.

These two additionally requirements stemming from the functional safety are not in the scope of this paper and will therefore not be considered in the following.

3.3 System Design

The considerations so far motivate to model the entire traffic system as a controlled system composed by two components for

- the *plant component* consisting of four lanes and
- the *controller component* calculating the duration of the green times $t_{NorthSouth}$ and $t_{EastWest}$ by an *explicit computation rule* from given numbers of the traffic members provides by the plant component.

Finally, we describe the interfaces. The interfaces between the plant and the controller component are given by six real values: four *numbers of vehicles in the waiting queues* n_{North} , n_{East} , n_{South} , and n_{West} and the two green phase values $t_{NorthSouth}$ and $t_{EastWest}$.

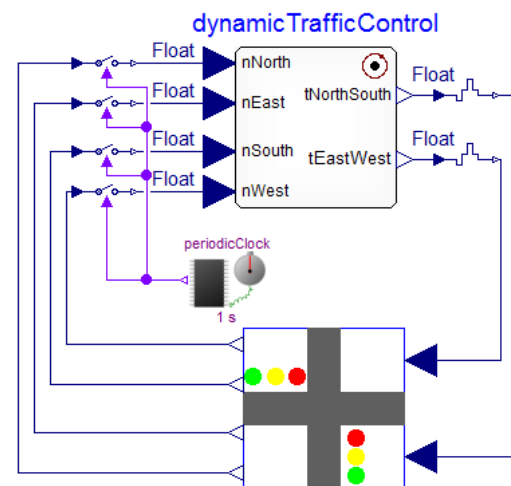


Figure 6: The composition of the system in Modelica. The symbol above right in the controller indicates the atomic execution behavior of the controller component.

The definition of the components and its interfaces, modeled in Modelica see Figure 6, is the first fundamental design decision, see the library SAFEDIS-CRETECONTROL in [11].

3.4 Module Design and Implementation

3.4.1 Component Plant

The component plant simulates an intersection of two roads, which runs in a north-south and an east-west direction, see again Figure 4. The four waiting queues are named by the facing directions North, East, South and West. We suppose a simple growth model for the population of the four lanes

$$\dot{n} = \begin{cases} c - c_{Out} & \text{if corresponding lane has green} \\ c & \text{else,} \end{cases} \quad (4)$$

where $n \geq 0$ denotes $n_{North,East, South}$ and n_{West} and $c \geq 0$ represent the *uniform growth constant*, and c_{Out} the additionally *decay constant of the waiting queues* in the green phase of the corresponding lane representing the number of vehicles passing the intersection.

The two opposite lanes are governed by two opposing traffic lights with the same signal sequence. Note that we neglect in the following the modeling of the yellow phase and it holds for the green time phases

$$t_{NorthSouth} + t_{EastWest} = t_{Clock}. \quad (5)$$

3.4.2 Component Controller

The component controller realizes roughly speaking a mapping from \mathbb{R}^4 to \mathbb{R}^2 fulfilling the requirements (1) and (2) - consequently, there exists an infinite number of implementations!

A very simple first strategy to fulfill the requirements is distribute the available time t_{Clock} equally to both green phases

$$t_{NorthSouth} = t_{EastWest} = t_{Clock}/2, \quad (6)$$

see also Figure 7 for the implementation in Modelica.

We initialize the component controller with red lights for both directions.

3.5 Module Integration and Test

In this phase, we validate the module designs and their implementations by so-called Model-in-the-loop simulations before we move on to the next development phase. Therefore, we analyse given use cases and test the controll algorithm by virtual integration.

We set for the module test phase the following general parameters:

- the minimim green time $t_{Min} = 10[s]$,
- the circulation time $t_{Clock} = 150[s]$, and

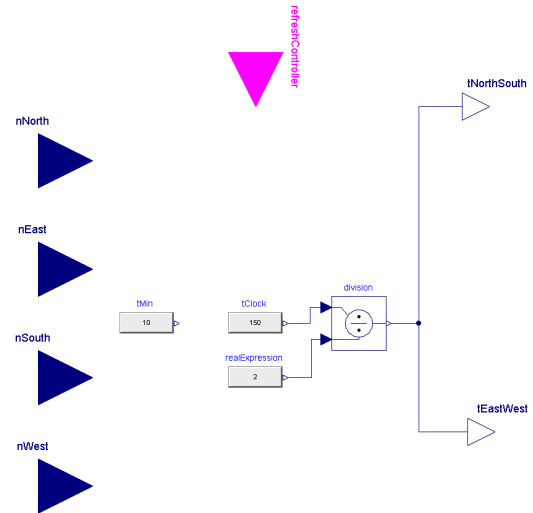


Figure 7: The implementation of the equations (6) for the symmetric strategy in Modelica.

- the initial numbers of vehicles in the waiting queues $n_{North} = n_{South} = 100[1]$ and $n_{East} = n_{West} = 50[1]$.

3.5.1 First Use Case: Equally Busy Lanes

In this use case, we assume that both roads *north-south* and *east-west* are equally frequented and chose the following use case specific parameters

- the growth constants $c_{EastWest} = c_{NorthSouth} = 1 \left[\frac{1}{s} \right]$ and
- decay constant of the waiting queues $c_{Out} = 2.2 \left[\frac{1}{s} \right]$.

Because it holds for the growth and decay constants

$$c_{EastWest} + c_{NorthSouth} \leq c_{Out}, \quad (7)$$

there may pass more vehicles through the intersection than new ones join in the waiting queues. We therefore expect a good controller to reduce the waiting queues over time.

Figure 8 shows the signal time plan corresponding to Figure 2. The evolution of vehicle values $n_{North} = n_{South}$ and $n_{East} = n_{West}$ in the waiting queues is given in the Figure 9.

This Model-in-the-loop simulations confirms the symmetric control strategy as expected. We therefore study a further asymmetric use case to test our first implementation.

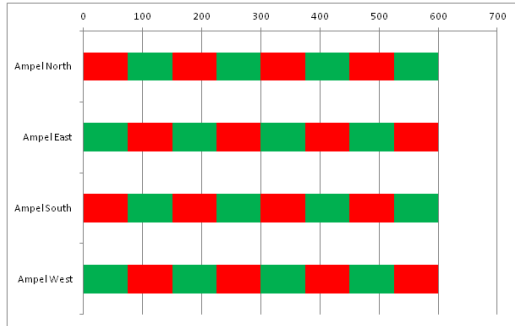


Figure 8: The signal time plan of the first use case.

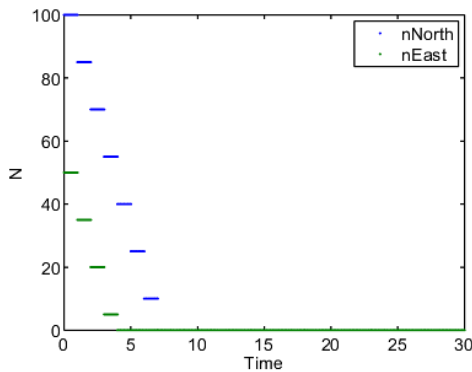


Figure 9: The evolution of vehicle values $n_{North} = n_{South}$ and $n_{East} = n_{West}$ in the waiting queues for the first use case.

3.5.2 Outlook: System Simulation

As an outlook, we mention here, that Modelica provides further tools for simulations like a full system simulation. The toolbox Modelica3D allows to visualize the full intersection. Figure 10 provides a picture of a movie produced by Modelica3D. For further details see [3, 2].

3.5.3 Second Use Case: Main and Secondary Road

We change the first use case only slightly and then simulate a scenario in which the north-south road is less traveled than the east-west road and assume the following parameters

- the growth constants $c_{EastWest} = 2 \left[\frac{1}{s} \right]$, $c_{NorthSouth} = 0.2 \left[\frac{1}{s} \right]$ and
- decay constant of the waiting queues $c_{Out} = 2.2 \left[\frac{1}{s} \right]$.

This time, as many vehicles arrive at the intersections as may pass through the intersection. We expect

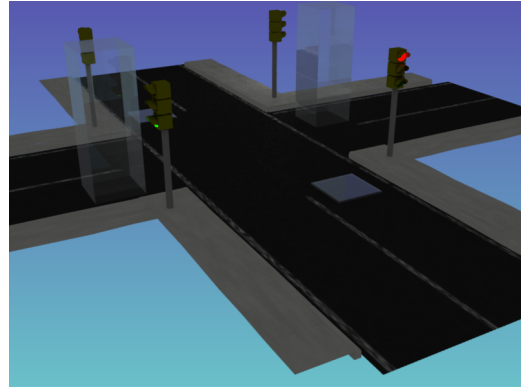


Figure 10: This picture of a movie produced by Modelica3D shows the behavior of the system example intersection. The waiting queues are visualized by boxes with heights depending on the length of the corresponding waiting queue.

a good controller not to increase the number of vehicles in the waiting queues.

The evolution of vehicle values $n_{North} = n_{South}$ and $n_{East} = n_{West}$ in the waiting queues is again given in the Figure 11.

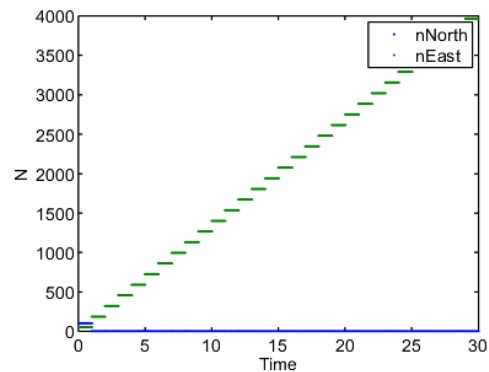


Figure 11: The evolution of vehicle values $n_{North} = n_{South}$ and $n_{East} = n_{West}$ in the waiting queues for the second use case.

This time, we observe that the North-south road drops to 0 and remains constant, whereas the East-west road linearly increases. The constant *green phase ratio*

$$t_{NorthSouth} : t_{EastWest} = 1 : 1 \quad (8)$$

obviously does not reflect the asymmetric *vehicle growth ratio*

$$c_{NorthSouth} : c_{EastWest} = 1 : 10 \quad (9)$$

good enough. This undesirable behavior motivates another requirement for the implementation of the controller algorithm.

3.5.4 Additional Requirement on System Level

We introduce two key indicators:

- the *ratio of the waiting queues* defined by

$$r_{wq} := (n_{North} + n_{South}) / (n_{East} + n_{West}) \quad (10)$$

and

- the *ratio of the green phases* given by

$$r_{gh} := t_{NorthSouth} / t_{EastWest}. \quad (11)$$

We assume that the longer the green phases, the more vehicles may pass the intersection. In the sense of the customers of the intersection, we therefore additionally require

$$r_{wq} \approx r_{gh}. \quad (12)$$

3.5.5 Module Design and Implementation for the alternative Controller

In this section, we develop a second, alternative controller, and solve therefore the system of equations (5) and (12) to fulfill mathematical exact the requirements. We define the *load distribution* for the two roads

$$\lambda := \frac{n_{East} + n_{West}}{n_{North} + n_{East} + n_{South} + n_{West}}. \quad (13)$$

The value $\lambda = 0 = 0\%$ reflects no traffic in east-west direction and consequently minimum green time for east-west and $\lambda = 1 = 100\%$ correspondingly for the other direction. Then, keeping in mind the minimum green time requirement (3), this yields to

$$\begin{aligned} t_{EastWest} &= \min(t_{Clock} - t_{Min}, \max(t_{Min}, \lambda \cdot t_{Clock})) \\ t_{NorthSouth} &= t_{Clock} - t_{EastWest}, \end{aligned} \quad (14)$$

see also Figure 12 for the implementation in in Mod- elica.

The so designed controller has the following desir- able properties:

$$\begin{aligned} \lambda = 0 : t_{EastWest} &= t_{Min}, \\ \lambda = 1 : t_{EastWest} &= t_{Clock} - t_{Min} \end{aligned} \quad (15)$$

with corresponding

$$t_{NorthSouth} = t_{Clock} - t_{EastWest}. \quad (16)$$

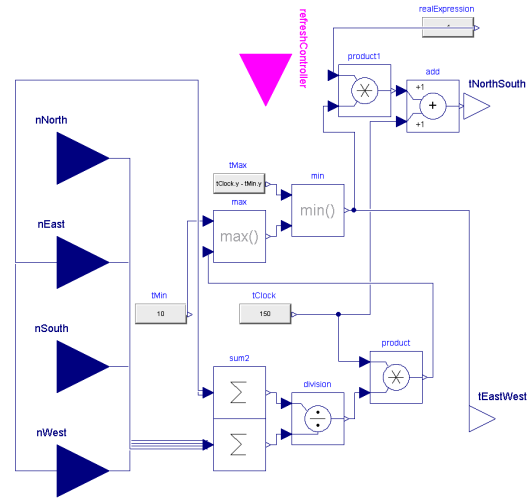


Figure 12: The implementation of the alternative con- trol algorithm given by (14) in Modelica.

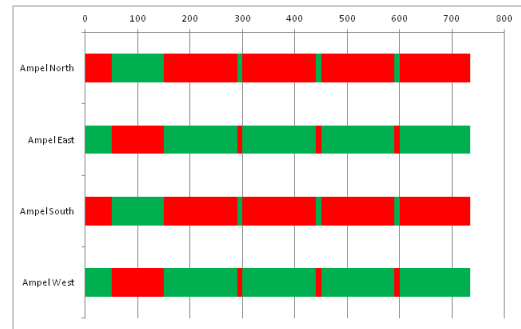


Figure 13: The signal time plan the alternative control algorithm given by (14) in Modelica.

3.5.6 Module Test and Integration of the alterna- tive Controller

Figure 13 shows the results of the Model-in-the-loop simulation of the second implementation of the controller for the second use case.

Figures 14 and 15 present the evolution of the num- ber of vehicles and the green phases.

The alternative controller responds to the asymmet- ric load much better. After a transient phase the ratio of the green phase 1 : 10 reflects the ratio of the loads 1 : 10 almost perfectly.

3.5.7 Regression of the First Use Case

Also the first use case can be controlled by the alter- nate controller. Although it produces, in contrast to the first controller, a small oscillation, but remain limited to vehicle values.

A simple validation shows that the second controller also produces the expected behaviour for the first use

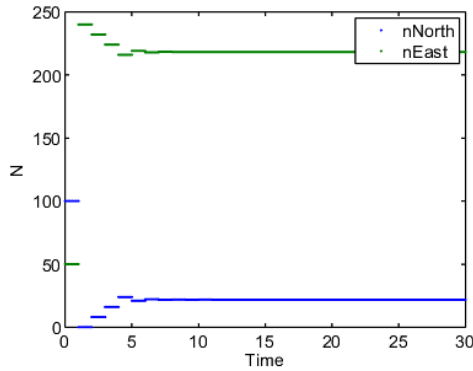


Figure 14: The evolution of the numbers of vehicles for the second use case *main and secondary road* with the second implementation of the controller.

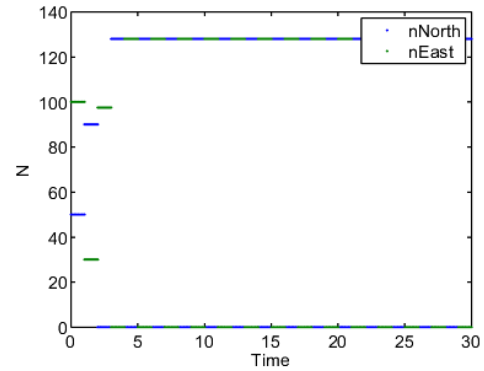


Figure 16: The evolution of the vehicle numbers for the first use case with the second controller, compare with Figure 9.

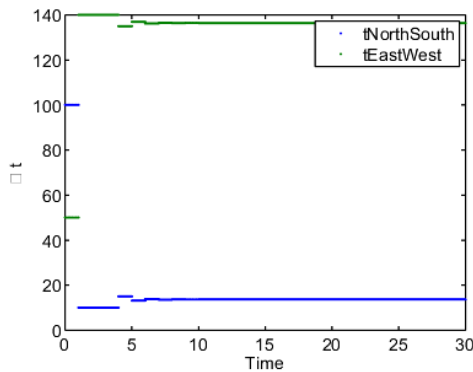


Figure 15: The evolution of the green phases for the second use case *main and secondary road* with the second implementation of the controller.



Figure 17: The PIC32 Starter KIT for the Processor-in-the-loop simulations showing the green LED representing the green traffic light.

case, see Figure 16.

3.6 System Integration and Test

In the last phase of the system development, we integrate the validated control algorithm in an evaluation board (MicroController with 80 MHz, 512KB Flash, 32KB RAM, USB) and development environment MPLAB Version 8.84 from Microchip Technology Inc., see Figure 17.

The presented approach differs from the method described in [8], where the control system is executed part on a PC, and part on a microcontroller board.

The relevant code fragment of the from Dymola produced file `dsmode1.c` can easily be identified for this specific controll development and integrated in the environment of the microcontroller code. This transformation can be performed automatically by a python script.

The Processor-in-the-loop simulations reflect in de-

tail the observed Model-in-the-loop results.

4 Conclusion

The application of the virtual integration has many advantages because it allows the observation of the behavior of a fully integrated system in an early development phase. Realistic tests in the early phase of development by virtual integration enables comprehensive evaluation of the interaction of a) functions, b) components, c) tools, and d) decision makers and allows a seamless, continuous development process. The method virtual integration allows therefore integration of new technologies and domains.

The following questions arises: How can we systematically identify other development-related interactions? This remains for future work.

References

- [1] Adolf-Peter Bröhl and Wolfgang Dröschel. *Das V-Modell. Der Standard in der Softwareentwicklung mit Praxisleitfaden*. Oldenbourg R. Verlag, September 1995. ISBN 978-3-348622-207-4.
- [2] C. Höger et al. Homepage Modelica3D, 2012. [Online; Status 24 May 2012].
- [3] C. Höger et al. Modelica3D - Platform Independent Simulation Visualization (submitted). In *Modelica Conferene 2012 & Conference Proceedings*.
- [4] R. Haberfellner, Olivier L. de Weck, E. Fricke, and S. Vössner. *Systems Engineering – Grundlagen und Anwendungen*. Orell Füssli Verlag, Zurich, 12th edition edition, January 2012. ISBN 978-3-85743-998-8.
- [5] Thomas Huckle and Stefan-Alexander Schneider. *Numerische Methoden: Eine Einführung für Informatiker, Naturwissenschaftler, Ingenieure und Mathematiker*. Springer, 2006.
- [6] IABG. V-Modell, 2004. [Online; Stand 24. Mai 2012].
- [7] Andreas Maier and Stefan-Alexander Schneider. Analyse des Einflusses der Co-Simulation bei der Modellintegration. *Tagungsband ASIM 2011*, 2011. ISBN 978-3-89967-733-1.
- [8] Marco Bonvinia, Filippo Donidab, Alberto Leva. Modelica as a design tool for hardware-in-the-loop simulation. Technical report, Dipartimento di Elettronica e Informazione, Politecnico di Milano, 2009.
- [9] Richard Harwell. Systems Engineering, A Way of Thinking, A Way of Doing Business, Enabling Organized Transition from Need to Product, 1997. [Online; August 1997].
- [10] Stefan-Alexander Schneider, B. Schick, and H. Palm. Virtualization, Integration and Simulation in the Context of Vehicle Systems Engineering. In *Embedded World 2012 Exhibition & Conference Proceedings*. Weka Fachmedien, 2012.
- [11] Stefan-Alexander Schneider, B. Thiele, and P. Mai. A Modelica Sub- and Superset for Safety-Relevant Control Applications (accepted). In *Modelica Conferene 2012 & Conference Proceedings*.
- [12] Tim Weilkiens. Die rolle des systems-engineerings.
- [13] Wikipedia. LISA+ — Wikipedia, Die freie Enzyklopädie, 2011. [Online; Stand 28. März 2012].
- [14] Wikipedia. Signalzeitenplan — Wikipedia, Die freie Enzyklopädie, 2011. [Online; Stand 28. März 2012].
- [15] Wikipedia. Systems Engineering — Wikipedia, the free encyclopedia, 2012. [Online; Status 13 May 2012].
- [16] Wikipedia. V-Modell — Wikipedia, Die freie Enzyklopädie, 2012. [Online; Stand 29. März 2012].
- [17] Wikipedia. V-Modell — Wikipedia, the free encyclopedia, 2012. [Online; Status 18. Juni 2012].

Acknowledgments

The authors hereby thank C. Höger from TU Berlin for the animation of the intersection in Figure 10 with Modelica3D and last but not least B. Thiele from DLR for the modeling of the control algorithms, e.g., in Figure 6 using the library SAFEDISCRETECONTROL, see [11].

Translating Modelica to HDL: An Automated Design Flow for FPGA-based Real-Time Hardware-in-the-Loop Simulations

Christian Köllner
FZI Forschungszentrum
Informatik
Haid-und-Neu-Str. 10-14
76131 Karlsruhe
koellner@fzi.de

Torsten Blochwitz
ITI GmbH
Webergasse 1
01067 Dresden
blochwitz@itisim.com

Thomas Hodrius
SET GmbH
August-Braun-Straße 1
88239 Wangen/Allgäu
hodrius@smart-e-tech.de

Abstract

Advances in the development of electric vehicles challenge existing test methodologies and tools. In particular, hardware-in-the-loop test rigs to verify electric motor controllers require real-time drivetrain emulation with response times in the order of one microsecond. Field-programmable gate arrays can fulfill these requirements due to their high parallelism and the possibility to realize efficient and predictable I/O interfaces. We present an integrated methodology which translates Modelica models to VHDL hardware designs. Our methodology combines well-engineered algorithms from Modelica compilation and high-level synthesis for hardware. We demonstrate its capabilities using the example of a DC motor which was synthesized and implemented on a Xilinx Virtex-5 device.

Keywords: FPGA; High-level synthesis; VHDL; Hardware-in-the-Loop; Real-time

1 Introduction

Recent movement towards electric vehicles imposes new challenges on the development of drivetrains. Especially the verification of electric motor controllers (EMCs) using the hardware-in-the-loop (HiL) test methodology requires real-time simulation of the functional environment with low latencies. An EMC is an integrated device, consisting of an electronic control unit (ECU) and a power stage. The ECU implements current, acceleration and/or speed control and safety functions whereas the power stage generates the motor currents. The test rig wires the EMC to an emulator, as shown in Figure 1. An electric motor emulator (EME) emulates an electrical motor under real conditions, including position feedback and other sensor signals. If needed, a power stage recreates the original currents and voltages.

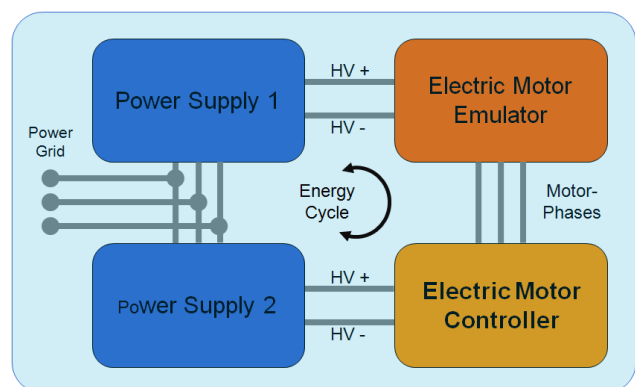


Figure 1: EMC test bed schematic

Due to the dynamic electric behavior of the motor, the model iteration rate has to be in the order of one microsecond. Since such real-time requirements are hard to meet using software solutions, HiL emulators of electric machines typically involve a field-programmable gate array (FPGA) which carries out time-critical computations. FPGAs are highly parallel reconfigurable hardware circuits which are well-suited for high-performance real-time computations. However, their programming model is fundamentally different from general-purpose computing. This fact makes current modeling environments lack an integrated flow from model to hardware. Although Modelica has proven to be an effective language for describing electric hybrid drivetrains [1], there is currently no tool support for compiling Modelica to FPGAs.

Our contribution tries to close this gap. We propose an integrated methodology for compiling Modelica models to an FPGA configuration. The implementation is realized and validated using SimulationX. Our approach combines well-known methodologies from both differential-algebraic equation (DAE) processing and high-level synthesis (HLS). We employ inline integration to obtain a compact calculation rule which can be efficiently mapped to

hardware. Moreover, we incorporate parametrizable circuit templates (so-called IP cores) to solve common subproblems during the mapping process.

Our paper is organized as follows: Section 2 investigates related work from commercial and academic perspective. Section 3 gives a short explanation of FPGA functionality and the programming model. Based on the specifics of FPGA operation, section 4 states the requirements to achieve an integrated, automated design flow from model to hardware. Section 5 explains these implications on model entry. In section 6, we discuss the overall design flow from Modelica to hardware. Section 7 presents the characteristics of an exemplary direct current (DC) motor model which was translated to hardware. Finally, section 8 concludes the paper and gives an outlook to future work.

2 State of the Art

Electric motor controllers used in automation and automotive applications combine controller and power stages in one device. Testing and verifying EMCs in an HiL environment is challenging, since the behavior of the electric motor must be rebuilt true to original. Otherwise, the EMC would diagnose a malfunction and enter failure mode. The interface between the EMC and HiL system can be realized on a mechanical, electric power, or signal level [2].

On the mechanical level, the original electric motor is connected to the EMC. Another motor is flanged and applies the mechanical load, computed online by a simulation model. Such dynamometer test stands (as shown in Figure 2) are expensive to build, hard to control, and not flexible in usage.

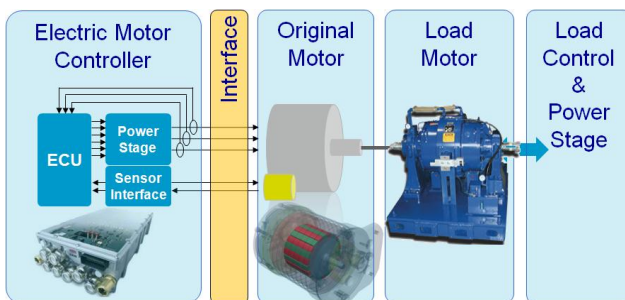


Figure 2: Dynamometer test stand

Interfacing on the signal level requires cutting the connection between the controller and power stage. This “cracked ECU” approach requires knowledge of controller internals. The behavior of the electric motor and its load is computed by a fast microprocessor or an FPGA device. The computed current-sensing signals are fed back to the ECU along with other

simulated sensor signals (shown in Figure 3). This approach excludes the power stage from test and verification.

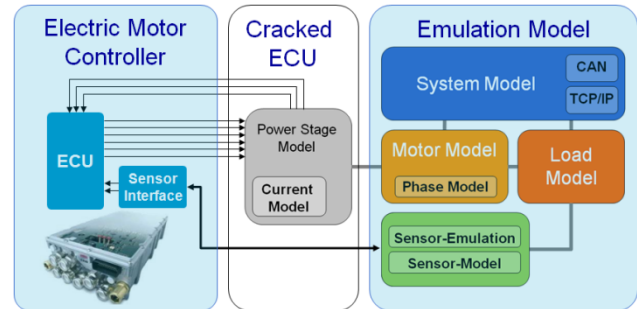


Figure 3: Cracked ECU test bed

When interfacing at the electric power level, the electric current is generated by special power electronics and fed back to the power stage of the unit under test. This methodology is referred to as Power Hardware-in-the-Loop (P-HiL). The SET EME realizes this methodology, reproducing proper power loads [3] without rotating parts (see Figure 4). The interface to the EMC is identical to the real motor. It consists of the motor phases and position sensor signals (e.g. resolver), if needed. Its applications vary from small servo controls with less than 100 W to electric power trains with several 100 kW. A wide range of motor types and rotor position interfaces is supported.

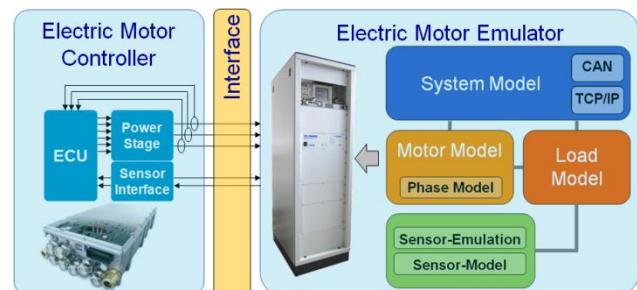


Figure 4: Electric motor emulator test bed

To achieve realistic emulation behavior, high switching frequencies of the EME power amplifiers are needed. This is especially important when operating at high rotational speeds and to emulate dynamic behavior, such as speed ramps. Hence, for these use cases special power amplifiers with application-dependent switching frequencies up to 800 kHz are deployed. Controlling the power amplifier requires input/computation/output latencies of 1.25 μ s.

Both the cracked ECU approach and P-HiL typically rely on FPGA-based implementations of the motor simulation. In absence of a suitable toolchain these models are commonly coded by hand, using a hardware description language (HDL). Examples

include a commercial model of inverter and permanent magnet synchronous machine (PMSM) [4], a DC motor [5], a squirrel-cage induction machine [6] and a generic implementation which covers an exhaustive set of AC motor types [7]. Yet, there is no general agreement on the type of arithmetic: most models incorporate fixed point arithmetic [5-7] whereas one contribution relies on floating point [4]. The development of such models is generally error-prone and time-consuming, especially if complex models (e.g. a nonlinear model of synchronous motors) or detailed drivetrains, including clutches and rigid end stops, must be realized.

In reference [8], HDL Coder from The Mathworks was used to implement a Simulink DC motor model on an FPGA. This toolchain is restricted to Simulink models without continuous states. User interactions and reformulation of the model are necessary to achieve a fast and synthesizable FPGA design. A similar approach is presented in [9]. The authors create a Matlab/Simulink model of a permanent magnet synchronous machine using the Xilinx System Generator (XSG) blockset. Again, the methodology requires the engineer to model at the hardware level. Reference [10] presents an approach to generate fixed point code from Modelica. It is capable of exporting Mittrion-C code for FPGA applications, but no details are given on how the transformation towards an FPGA design works, and no FPGA implementation is presented.

3 FPGA Fundamentals

3.1 Overview

An FPGA is an integrated digital circuit whose functionality is programmable after manufacturing. To achieve programmability, FPGAs generally provide configurable combinatorial logic blocks and memory elements. These can be wired in a large variety of ways. By combining both primitives – logic and memory – it is theoretically possible to recreate any digital circuit. Recent FPGAs are computationally equivalent to roughly 20 million logic gates. Most devices provide additional built-in macro cells for frequent tasks, such as hardware multipliers and static RAM.

3.2 Programming FPGAs

In most cases, a hardware description language (HDL), such as VHDL and Verilog is used to describe the intended digital circuit. Vendor-specific toolchains transform the described design into a

netlist representation, map it to device primitives, optimize the geometric placement of that mapping and finally produce a programming file which configures the FPGA.

HDLs also define control-flow statements, which in fact turn them into general-purpose programming languages. However, these constructs are primarily intended for simulation/verification purposes and are mostly not supported for circuit modeling. A HDL description is said to be synthesizable, if it is possible to represent it by a functionally equivalent netlist. Therefore, synthesizability is a mandatory prerequisite to FPGA configuration. Particularly, analog-mixed signal extensions of VHDL (VHDL-AMS [11]) are generally not synthesizable.

3.3 Example

The following example is kept in VHDL and illustrates the impact of a specific notation on the synthesized circuit. Assume that we want to transform the following computation into a digital circuit:

$$r := a \cdot b + c \cdot d$$

If we encode all operands using a fixed point representation, there is a straightforward VHDL translation of the given calculation rule:

$$r \leftarrow a * b + c * d;$$

This implementation implicitly prescribes a combinatorial, fully-spatial realization. Synthesis infers a circuit which consists of two multipliers and one adder. Although this is the fastest possible realization, it may miss a design goal: Embedded in a synchronous design, this circuit may drop the achievable clock rate because of its combinatorial path. This can be avoided by buffering multiplication results in intermediate registers. If we need to save FPGA resources, a longer computation time might be acceptable. In this case, the calculation can be described as finite state machine (FSM):

```

Compute: process (Clk)
begin
  if (rising_edge(Clk)) then
    case state is
      when Mul1 => tmp1 <= A * B;
                    state <= Mul2;
      when Mul2 => tmp2 <= tmp1;
                    tmp1 <= C * D;
                    state <= Add;
      when Add => R <= tmp1 + tmp2;
                    state <= Mul1;
    end case;
  end if;
end process;

```

This implementation spreads the computation across three clock cycles. Since at most one multiplication happens per clock step, synthesis will share

resources: the novel circuit requires only one multiplier instead of two.

Changing the computation to floating point arithmetic requires the designer to use either special libraries or to interface the design with an IP core. IP cores are pre-built circuit templates with well-defined functionality which are either supplied by the device manufacturer or third-party vendors. This option usually provides better performance and detailed hardware tuning parameters. IP cores are also available for advanced mathematical operators, such as division, square-root and trigonometry.

High-level synthesis (HLS) is a field of research which addresses automated transformation of formal behavioral descriptions (mostly C/C-like programming languages) to hardware [12]. The transformation is constrained by requirements, such as resource consumption and time. Despite commercial tools are available, their success is limited. This is not only due to their high asset costs but also due to the user's uncertainty with respect to the quality of results [13]. Their effectiveness varies strongly with problem domain and coding style. Our contribution exploits the ideas of high-level synthesis. By tailoring its methodologies to the specific area of physics simulation we get a domain-specific approach which is able to meet our resource and timing requirements.

4 Requirements

The intended application imposes several implications on the chosen approach and equation processing. The following subsections discuss them in more detail.

4.1 Inline integration

Typical code generation from Modelica relies on a software infrastructure which distinguishes solver and model. The solver is in control of the overall simulation and employs callback functions to transfer control to the model-specific evaluation of derivatives. A tight interaction with strong data dependencies connects the solver and model components. This interaction is entirely time-multiplexed, exposing only little potential to parallelize [14]. Establishing a spatial distinction between solver and model on the FPGA would produce hardly any benefit. Thus, it is preferable to synthesize a self-contained calculation rule which encompasses the overall computation to carry out one integration step. This technique is called inline integration [15].

4.2 Real-time execution

During real-time computation, two conditions must be fulfilled: First, the computation time to perform a single integration step must be bounded and predictable. Second, the integration step size must have a lower bound. Since data acquisition and output of an HiL emulator usually happen at a fixed sample rate, it is even desirable to employ a fixed-step integration method.

Moreover, Modelica events must be used with care. Due to the fixed step size, the precise time instance of state events cannot be localized. Events are shifted to the end of the current integration step. In our case, this should not lead to problems because the step size used on a FPGA device is small compared to common processor-based HiL systems.

At event instances, a Modelica simulator carries out event iteration. The model is recomputed at the same time instance until discrete variables do not change anymore. The number of necessary event iteration steps cannot be predicted. Hence, the real-time condition might be violated. For that reason the model should be built in such a way that avoids event iterations. The Modelica compiler should recognize if the model requires event iterations (e.g. due to algebraic loops over discrete variables) and inform the user.

Implicit integration methods as well as algebraic constraints can necessitate the solution of non-linear systems of equations during simulation. Since such systems are usually solved by numerical methods, it is not guaranteed that the solution algorithm converges within a bounded number of iterations. Therefore, non-linear systems of equations should be avoided by the model. Ultimately, Modelica allows for embedding arbitrarily complex algorithms into any computation. It is the designer's duty to ensure that they have bounded execution times.

4.3 Choice of arithmetic

PC-based simulations usually rely on IEEE 754 floating point data types. Although this type of arithmetic can be implemented on an FPGA, it has weaker performance and higher resource consumption compared to equally-sized fixed point data. The situation changes if an adequate fixed point representation would require disproportionately large word sizes. FPGAs support "uncommon" word lengths (which are not powers of two). An appropriate synthesis flow should exploit these facts and support both – possibly mixed – floating point and fixed point arithmetic operators.

4.4 Sustaining domain-specific knowledge

A key challenge is to identify the level of abstraction at which a preprocessed model should be handed over to the hardware-centric synthesis flow. Physical computations involve many subproblems which can be directly mapped to IP cores. Examples are mathematical operators, such as sine/cosine, square-root and the absolute value function. Calls to such functions should be preserved in order to give the synthesis flow a chance to adopt dedicated hardware components. Another example is the solution of linear equation systems, which is necessary to simulate models with algebraic loops. In the past, numerous high performance linear solvers for FPGAs were developed [16-19]. To enable their usage, model pre-processing should keep linear systems instead of inserting a specific solver algorithm.

4.5 Minimizing computation effort

Compiler optimizations, such as common sub-expression elimination and exploiting algebraic identities are particularly important when targeting FPGAs. Device resources are limited, and each additional operation will affect either performance or area. Conversely, the slimmer design will fit on the smaller and cheaper device. Although it is possible to generate FPGA solvers for linear or nonlinear equation systems, avoiding such systems helps to keep the design compact.

5 FPGA-Aware Modeling

As implied by the special capabilities and limitations of FPGAs, the user should adhere to certain modeling guidelines when designing models for FPGA execution. Violating them can cause the translation to fail or lead to bloated hardware designs. We implemented a Modelica library prototype which contains frequently used elements for modeling electrically driven drivetrains and takes these aspects into account. Using this library and considering some modeling guidelines will lead to synthesizable designs faster than using the general purpose Modelica Standard Library or the SimulationX libraries. Figure 5 shows the structure of the library.

Special considerations were necessary for the dry friction model. Real-time motor emulation requires a robust friction model that reproduces correct stiction behavior. Usage of the friction element should neither result in a combined discrete continuous system

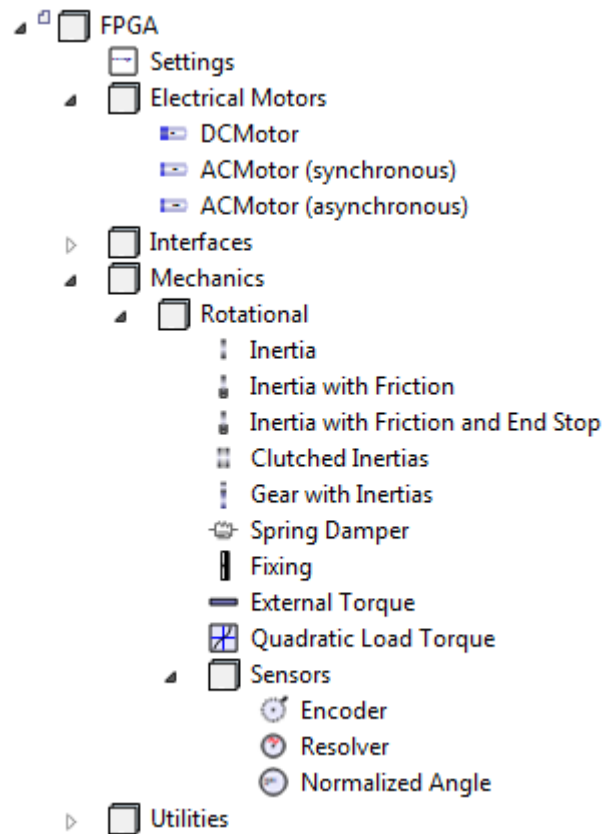


Figure 5: Screenshot of the library structure

of equations nor cause event iteration. By combining friction behavior with inertia, the resulting friction torque and the new discrete state can be computed explicitly. The solution of a system of equations and event iteration become obsolete. This approach is used by the library elements “Inertia with Friction” and “Clutched Inertias.”

Further systems of equations can be avoided, if some modeling guidelines are obeyed. For example, an inertia element should be placed between elements which introduce a torque to the system (spring dampers, motors, loads). Inertia elements should not be strung together. These rules do not restrict the model features which can be represented by the library. Only the way in which models are to be built up is slightly constrained. If the rules are violated and systems of equations persist, the Modelica compiler generates appropriate warnings.

6 Compilation and Synthesis

Figure 6 illustrates the overall design flow which is implemented by our software prototype. The following subsections explain the procedure step-by-step.

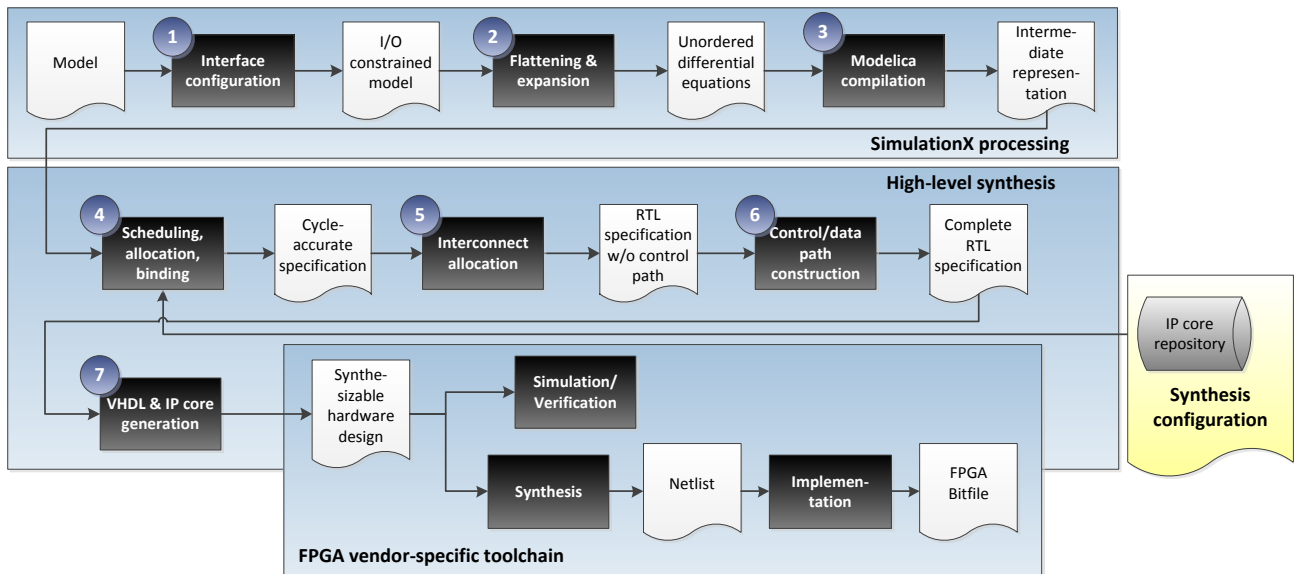


Figure 6: Overall model compilation and synthesis flow

6.1 Preparation of the model

First, the interface of the model is to be specified. The user selects inputs, outputs and parameters which shall be available on the FPGA. Inputs, outputs and parameters will become VHDL ports of the generated hardware design unit.

6.2 Modelica compilation

Most stages of the compilation process are not specific to FPGA code generation. Some steps after flattening (step 2) of the Modelica model are specific according to the requirements of Section 4. In order to reduce the complexity of the resulting VHDL code, loops of known and constant range are unrolled, and equations of higher dimension are expanded. Furthermore, equations and variables which do not influence the selected model outputs are removed. Functions are inlined since function calls would bloat the hardware by requiring an execution stack.

Since state events cannot be precisely located anyway, all conditions are covered implicitly by the `noEvent(...)` function. Algebraic loops containing discrete variables would require event iteration. This case is detected by the SimulationX Modelica compiler which displays an appropriate message. The integration formulas for computing the values of continuous states from their derivatives are introduced in an early stage of the compilation process. This enables symbolic simplifications on these parts of the algorithm too. We use Euler's forward integrations method, which is a good compromise between computational effort and stability.

The SimulationX compiler produces either C code or a bytecode representation for simulation. We extended its capabilities to generate an XML-based assembler-like intermediate representation to be processed by the FPGA-centric tooling. The instruction set was chosen to match hardware capabilities. For example, op-codes for common mathematical operators exist which allow fixed point and floating point operands of arbitrary sizes. The resulting behavioral description basically contains two procedures:

- Initialization part
- Iteration part

The initialization part is an algorithm which computes initial variable values from all model parameters. It may also perform some non-trivial computation, such as iteration to find consistent state values. Since it is executed only once (at the beginning of the simulation), it is not time critical. The iteration part contains the actual computation which is performed during simulation. It is a function of model inputs and state, transforming those quantities into output and new state. This algorithm gets iterated for each time step and therefore must have a predictable and bounded execution time.

6.3 Scheduling

When mapping an algorithm to hardware, three fundamental tasks need to be distinguished:

- Scheduling assigns execution time (i.e. clock tick) to each instruction.
- Allocation determines which hardware functional units (FUs) to instantiate and in which quantities. For each instruction there must be at least one FU which can execute it.

- Binding assigns each instruction to a FU. It must ensure that no two instructions are assigned to the same FU at the same time. It should also account for interconnection costs which are induced by its choice.

Superscalar processors perform scheduling and binding dynamically (allocation is determined by manufacturing). They analyze the incoming instruction stream for data dependencies and schedule them automatically. A tremendous amount of logic is required to achieve such functionality. Recreating superscalarity on an FPGA is not a viable option. Instead, a static schedule is pre-computed. Another advantage is that execution time is completely predictable.

Our prototype employs the force-directed scheduling algorithm (FDS, [20]). FDS is a time-constrained approach which exploits instruction-level parallelism. Its input is a control-/data-flow graph (CDFG) and a time constraint. Upon success, it returns a schedule which heuristically minimizes the amount of required FUs. Generous time constraints lead to fewer FUs and therefore reduce resource consumption. Figure 7 shows the scheduled CDFG of a DC motor model. The model itself will be introduced in Section 7. Each rectangle depicts a variable/constant load/store instruction whereas each circle depicts an arithmetic operation. In the given example, multiplication was configured to last three cycles, addition/subtraction two cycles.

6.4 Allocation and binding

Allocation and binding are downstream stages to scheduling. The schedule determines the minimum amount of FU instances of each kind which are required. It does not prescribe which instance will actually execute a specific instruction. Binding multiple staggered instructions to the same FU is called resource sharing. Obviously, sharing is desirable, since it helps to reduce the area of the overall hardware design. On the negative, it can lead to performance degradation. Input multiplexers will be necessary to select from different operands. They increase the combinatorial delay and may affect the clock rate. If the operand sources get placed at far-off chip locations, routing delays will further drop the clock rate.

We employ a heuristic to tackle the problem. Our algorithm sequentially assigns each instruction to an FU by either allocating a new FU instance or reusing a previously allocated one. In case of reuse, assignments that reuse existing interconnect are preferred. If reusing any previously allocated FU would require

overly large multiplexers, a new FU is allocated instead.

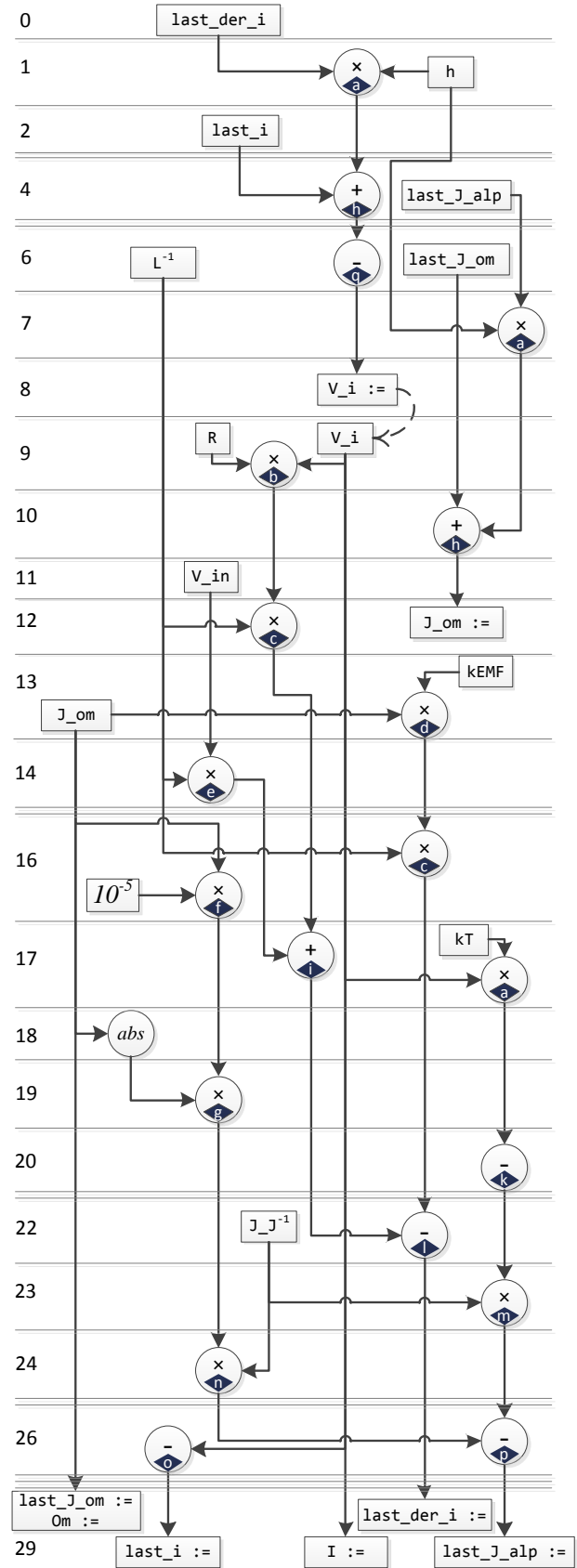


Figure 7: Scheduled and bound CDFG of a DC motor with quadratic friction

The result of allocation/binding the DC motor CDFG is shown in Figure 7: Characters inside diamonds enumerate the FU instances which the operations were mapped to. The operating point was set to spare resource sharing in favor of performance. Moreover, the outcome suggests that the binding procedure was able to identify the most economic candidates for resource sharing: The multiplications in control steps 1 and 7 are mapped to the same hardware multiplier. This is reasonable, since both operations share the common operand h .

The set of instantiable FUs is provided by an IP core repository. It must hold an according FU type for each kind of instruction. The repository is assembled from hand-written cores as well as vendor-specific IP cores. The latter are shipped with the FPGA toolchain and provide off-the-shelf implementations of complex arithmetic units, such as floating point operators, trigonometric operators and square-root.

6.5 Interconnect allocation

Once the complete instruction stream is scheduled and bound to appropriate FU instances, an interconnect network is constructed. It is responsible for routing operational results to their target FUs. The schedule may also require the network to buffer intermediate results. This happens if a result is not processed within the same clock step it was produced. Thus, the interconnect network is composed of multiplexers and flip-flops.

We developed an incremental merging heuristic which considers both register count and multiplexer size. An initial solution is constructed by assigning each instruction outcome to an individual storage register. Afterwards, register pairs are iteratively selected and merged whereby the merging decisions try to balance the multiplexer sizes of the overall interconnect structure.

6.6 Control path construction

The control path is a hardware unit which conducts the temporal interaction of all data path components. This includes asserting handshake signals and setting an input selection for each multiplexer. After the scheduling, allocation/binding and interconnect allocation steps have been completed, the control path is completely specified in its behavior. It just needs to be expressed by an explicit implementation. In the scope of this contribution, an FSM representation was chosen. Each control step of the schedule constitutes one state. A VHDL process steps the state forward with each rising clock edge.

Another combinatorial process computes appropriate settings for handshake signals and multiplexers, based on the current state. FSM descriptions are recognized by FPGA synthesis tools. These try to infer an optimal hardware representation for the given FSM. To support optimal inference, we represent the state variable using a VHDL enumeration data type. This gives VHDL synthesis a chance to choose an optimal state encoding [21].

6.7 Source code generation

The generated design involves VHDL source code, but also parameterization scripts for vendor-specific IP cores which were instantiated from the IP core repository. Although our approach is conceptually independent of device technology, the generated design is technology-dependent if it involves vendor-specific IP cores. So far, Xilinx FPGAs are supported.

7 Results

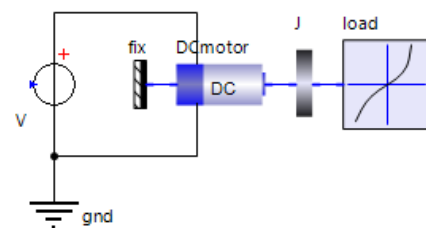


Figure 8: Sample model

We demonstrate the transformation process using the model of a DC motor (Figure 8). The motor is connected to an inertia and a load torque with quadratic dependency on speed. This is the typical behavior of a fan. The voltage at the voltage source ($V.v$) is used as input, current ($V.i$) and motor speed ($J.om$) are the outputs.

The generated VHDL code is synthesizable on an FPGA. All `Real` variables of the Modelica model are represented by fixed point numbers with 32 bits precision at inputs and outputs. Intermediate results are processed at higher precision. The proportioning into integral and fractional part was done individually for each quantity, with respect to its range of values. Figure 9 compares the output values of the VHDL code to the simulation results, using the Euler forward method and a step size of $1 \mu\text{s}$. The motor is fed by a voltage jump of 12 V. The simulation results are reproduced with sufficient accuracy. Minor deviations are caused by the fixed point representation of the variables in VHDL.

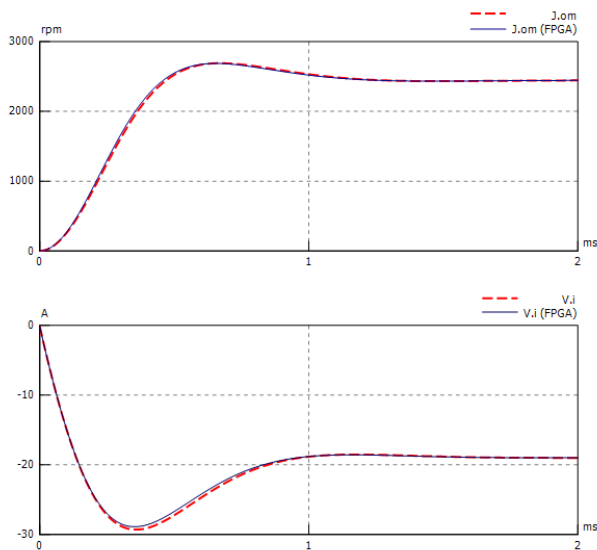


Figure 9: Simulation results (red) and FPGA results (blue)

To achieve synchronized data transfer, the design unit is equipped with additional handshake signals. These signals control initialization and model evaluation. Figure 10 shows the basic structure of the resulting hardware design unit. Model initialization and evaluation are separated into two individual FSMs which share a register bank. Asserting the `Init` signal causes the initialization procedure to capture and preprocess all parameters. This includes precomputing the reciprocals of moment of inertia (J_J) and rotor coil inductivity (L). Since division is a costly hardware operation, this step improves runtime performance.

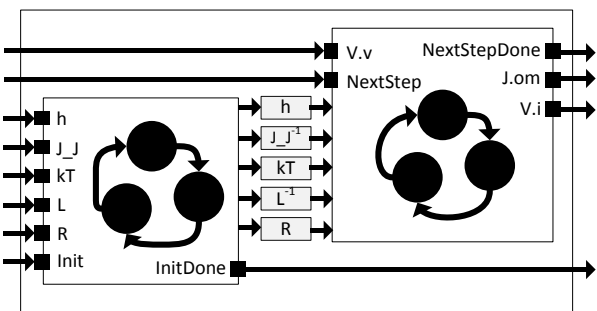


Figure 10: Architectural overview of the generated hardware design unit

Figure 11 shows the interplay of all handshake signals. Once the initialization is complete, model evaluation is controlled by the signals `NextStep` and `NextStepDone`. As noted in Section 3, the latencies of arithmetic operators are design parameters and affect computation time, clock rate and chip area. Although low latencies reduce the overall computation time, this usually comes at the cost of clock rate.

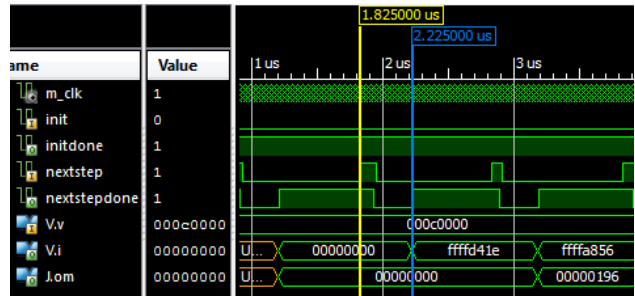


Figure 11: Initialization and runtime behavior of the design unit

The goal was integrate the generated design into SET’s EME hardware. Due to the hardware requirements, the design must achieve a clock rate of 100 MHz on a Virtex-5 LX110 device and complete any model evaluation within 1 μ s. Consequently, the schedule of the overall computation (an example is given in Figure 7) must not exceed 100 clock cycles. Using three different configurations, we generated corresponding design variants.

Table 1: Characteristics of the generated designs

L_{mul}	L_{add}	L_{tot}	<i>Slice usage</i>	F_{max} (MHz)
1	1	17	5%	89
3	2	30	6%	105
9	3	43	6%	102

Table 1 summarizes the characteristics of the generated designs. The columns depict, from left to right: 32×32 bit multiplication latency, 32 bit addition latency, schedule length of model evaluation, slice usage and maximum achievable clock rate after placing and routing the design on the target device. Slice usage is an approximate measure of the chip area which is consumed by the hardware design.

Although the first variant provides the fastest computation time, it does not reach the target frequency of 100 MHz. The remaining two alternatives are both viable. However, the second option is superior compared to the third one. It provides an overall input/output latency of 400 ns at 100 MHz, including handshake-induced wait cycles. This is more than sufficient to meet the requirement of 1 μ s.

8 Conclusions and Outlook

The toolchain approach described in this document will allow the efficient realization of flexible electric motor emulators. The combined model of motor and drivetrain is built using the FPGA-aware Modelica library. The resulting model is automatically transformed to an FPGA design. The FPGA controls the

EME hardware. Although the computation needed to accomplish a DC motor simulation is manageable, its hardware implementation introduces many new degrees of freedom: architecture, scheduling, resource allocation and binding, parameterization of arithmetic data types and corresponding hardware operators. Designing such hardware manually is a complex and time-consuming task. If the first draft does not meet the design goals, alternative implementations need to be explored, multiplying the effort. This contribution will allow an EME operator to model an application using SimulationX and link it directly to the hardware – even with moderate FPGA knowledge.

One of the next steps in our joint research project is the semi-automatic determination of the optimum fixed point representation for the model variables. A compromise between accuracy and occupied FPGA resources is to be found. It is also conceivable to realize a hybrid approach which combines fixed and floating point arithmetic in a single model, based on cost/accuracy tradeoffs.

Another field is the convenient subdivision and numerically robust reconnection of sub models. This becomes eminent as soon as a complex model exceeds FPGA resources. In this case, slow sub models could be computed on a microprocessor, and only the fast parts run on the FPGA.

The presented work is not restricted to electric motor emulation. It would be highly interesting to evaluate it for implementing sophisticated control algorithms on FPGA devices, based on Modelica models.

9 Acknowledgment

The presented work was accomplished within the project SimCelerate, which is funded by the Federal German Ministry of Education and Research (grant no. 01M3196C).

References

- [1] Winkler D., Gühmann C. Hardware-in-the-Loop simulation of a hybrid electric vehicle using Modelica/Dymola. Yokohama, Japan: The 22nd International Battery, Hybrid and Fuel Cell Electric Vehicle Symposium & Exposition, Japan Automobile Research Institute, 2006
- [2] Köhl, S., Himmeler, A.: Anwendungen und Trends bei der HiL-Simulation. Simulation und Test in der Funktions- und Softwareentwicklung für die Automobil-elektronik II, expert verlag, Berlin, 2008, pp. 203-217
- [3] SET GmbH Echtzeit-Emulation beschleunigt die Entwicklung, Funktions- und Leistungstests von E-Motor-Steuergeräten, Markt&Technik Vol. 27, 2010-05
- [4] Liebau H., Jakoby H., Crepin, J.: HiL-Simulation elektrischer Fahrzeugantriebe. Automotive Engineering Partners, Vol. 2011-05
- [5] Zhou Y. J., Mei T. X., FPGA based real time simulation of electrical machines, Proc. 16th IFAC World Congress, 2005
- [6] Matar M., Irvani R., Massively parallel implementation of AC machine models for FPGA-based real-time simulation of electromagnetic transients, IEEE Transactions on Power Delivery, Vol. 26, No. 2, pp. 830-840 2011
- [7] Chen H., Sun S., Aliprantis D., Zambreno J., Dynamic simulation of electric machines on FPGA boards, Electric Machines and Drives Conference, 2009
- [8] Köllner C., Yao H., Müller-Glaser K. D.: Entwurfsmethodiken zur Echtzeitsimulation physikalisch motivierter Modelle auf FPGAs: Eine Fallstudie. Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen (MBMV), 2011.
- [9] Dufour C., Belanger J., Lapointe V., and Abourida S., "Real-time simulation on FPGA of a permanent magnet synchronous machine drive using a finite-element based model," Symposium on Power Electronics, Electrical Drives, Automation and Motion (SPEEDAM), 2008
- [10] Nordström U., López J. D., Elmqvist H., Automatic Fixed-point Code Generation for Modelica using Dymola, Proc. Intl. Modelica Conf., 2006
- [11] VHDL Analog and Mixed-Signal Extensions, IEEE Std. 1076.1-1999
- [12] Coussy P., Morawiec A. High-Level Synthesis: from Algorithm to Digital Circuit. Springer Netherlands, 2010.
- [13] Grant M., Smith G. High-Level Synthesis: Past, Present, and Future. Journal: IEEE Design and Test of Computers. Vol. 26, pp. 18-25, 2009.
- [14] Nyström K., Aronsson P., Fritzon P., Parallelization in Modelica, Proc. 4th Intl. Modelica Conf., 2005
- [15] Elmqvist H., Otter M. and Cellier F.E.: Inline Integration: A New Mixed Symbolic/Numeric Approach for Solving DAE Systems. Proc. ESM'95, European Simulation Multiconf., 1995.
- [16] Johnson J., Chagnon T., Vachranukunkiet P., Nagvajara P., Nwankpa C., Sparse LU Decomposition using FPGA, International Workshop on State-of-the-Art in Scientific and Parallel Computing (PARA), 2008
- [17] Daga V., Govindu G., Prasanna V., Gangadharalli S., Sridhar V., Floating-point based block LU decomposition on FPGAs, Proc. Intl. Conf. on Engineering Reconfigurable Systems, 2004
- [18] Gonzalez J., Núñez R. C. LAPACKrc: Fast linear algebra kernels/solvers for FPGA accelerators. Journal of Physics: Conference Series. 2009
- [19] Fischer T., Entwurf eines FPGA-Cores zur Simulationsbeschleunigung zeitkontinuierlicher Modelle im HiL Kontext. GI Fachtagung Echtzeit 2011 - Herausforderungen durch Echtzeitbetrieb, 2011
- [20] Paulin P. G., Knight J. P. Force-directed scheduling in automatic data path synthesis. Proc. 24th ACM/IEEE Design Automation Conf. (DAC), 1987
- [21] Xilinx, Inc. Synthesis and Simulation Design Guide. UG626 (v13.4), 2012

A Modelica Library for Real-Time Coordination Modeling

Uwe Pohlmann¹, Stefan Dziwok¹, Julian Suck¹, Boris Wolf¹, Chia Choon Loh², and Matthias Tichy³

¹Software Engineering Group, ²Control Engineering and Mechatronics Group,

Heinz Nixdorf Institute, University of Paderborn, Paderborn, Germany

[upohl | stefan.dziwok | jsuck | borisw | chia.choon.loh] @upb.de

³Department of Computer Science and Engineering, Chalmers | University of Gothenburg, Sweden
tichy@chalmers.se

Abstract

Increasingly, innovative functionality in embedded systems is realized by connecting previously autonomous embedded systems. This requires real-time communication and coordination between these connected systems. Modelica and the StateGraph2 library provide a good environment for modeling embedded systems including controllers and physics. However, it lacks appropriate support for modeling the communication and coordination part.

In this paper, we present an extension to the StateGraph2 library that enables modeling asynchronous and synchronous communication and rich real-time constraints. We illustrate our extension of the StateGraph2 library by modeling and simulating two miniature robots driving in a platoon.

Keywords: StateGraph2, Modelica Library, Coordination, Asynchronous Communication, Real-Time

1 Introduction

Embedded software is an important part of today's life. For example, there were about 30 embedded microprocessors per person in developed countries in 2008 and current cars include up to 70 electronic control units with about 1GB of software [4].

One reason for the increasing trend of embedded systems is the introduction of coordination between previously autonomous systems. As a result complex systems of systems arise to realize functionality which cannot be achieved by each system alone [12]. Again, the car industry is an example where vehicles communicate with other vehicles in order to extend the car's vision to areas obstructed by other vehicles [15]. This coordination requires an intensive communication between the systems under real-time constraints.

The embedded software is subject to very high qual-

ity requirements as often embedded systems are safety-critical systems where faults can result in severe consequences, e.g., injuries or loss of peoples' lives. Thus, faults of the system have to be avoided as much as possible. Currently, the rate of defects from mechanical parts decreases while the defect rate in electrical parts including software increases [4].

Therefore, appropriate validation and verification activities, e.g., simulation, have to be employed to detect and remove all faults. Model-driven development approaches allow to perform these activities already on the model level in early phases of development. Thus, on the one hand, a verification approach can exploit the abstraction provided by the model to improve the scalability and, on the other hand, verification can already be performed early in the process where no implementation yet exists.

Modelica is an object-oriented, declarative, multi-domain modeling language for describing and simulating models which represent physical behavior, the exchange of energy, signals, or other continuous-time interactions between system components as well as reactive, discrete-time behavior. Modelica uses the hybrid differential algebraic equation formalism as a sound mathematical representation. Furthermore, mature compilation and simulation environments for Modelica exist.

However, Modelica in version 3.2 and particularly the StateGraph2 library lack appropriate support for the sketched case of modeling the real-time coordination between autonomous systems as this coordination is often realized by communication using asynchronous messages and complex state-based behavior [12].

In this paper, we present a Modelica library for modeling communication under hard real-time constraints. Our library extends the StateGraph2 library by providing support for (1) synchronous and asynchronous communication and (2) rich modeling of real-time behavior.

These extensions are based on our previous work on the MECHATRONICUML modeling language [2] and ModelicaML [11].

In the next section, we present our running example. We discuss the limits of the StateGraph2 library with respect to this scenario in Section 3. Our extension to the StateGraph2 library is described in Section 4. We formally define our extension in Section 5. In Section 6, we present the Modelica model of our scenario using our library extensions. After a discussion of related work in Section 7, we conclude and give an outlook on future work in Section 8.

2 Running Example

This section presents our test platform for evaluating real-time coordination scenarios. We present a concrete real-time coordination scenario of a platoon drive as the running example for the paper.

2.1 Intelligent Miniature Robot BeBot

The test platform is a wheeled mobile robot known as BeBot [7]. It is a miniature mobile robot developed at Heinz Nixdorf Institute and has been used in various research projects, e.g., [8]. The BeBot is powered by two DC-motors with integrated encoder.

To use this mobile robot in a simulation environment, a model of the BeBot is developed in Dymola. Basically, the hardware model of the mobile robot can be categorized into three main groups. The first group consists of its casing and electrical circuit boards. All these components are modeled as a rigid body in Dymola. In addition, the shape model from the *MultiBody* library is used to visualize these components in the animation. The second group comprises the wheels of the robot. Under the assumption of pure rolling, these wheels are represented by a pair of wheels with a common axle whereby each wheel is individually controlled. The third group is made of two DC-motors. Each of these



Figure 1: Intelligent Miniature Robot BeBot

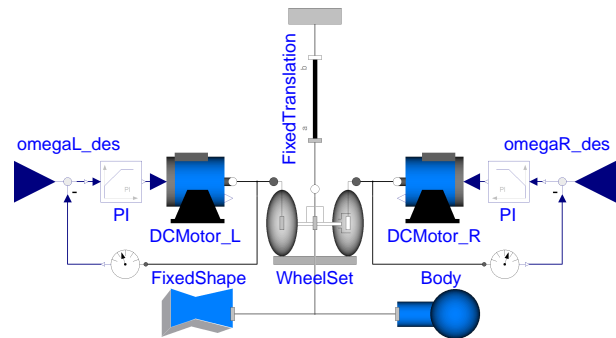


Figure 2: Model of BeBot Mobile Robot in Dymola

motors is represented using a model of a DC-motor. In this model, friction is taken into consideration to provide realistic behavior for the motor. As shown in Figure 2, these components are connected accordingly to create a simple model of the BeBot.

To control the movement of the mobile robot, the velocities of the wheels have to be controlled. Therefore, a speed controller is designed to control the rotation velocity of each wheel. The controller is a PI-controller with anti-wind-up function and it ensures that each wheel rotates at a desired velocity.

2.2 BeBot Platoon Scenario

The scenario consists of two BeBots (see Figure 3). They communicate wirelessly with each other and have a distance sensor at their front. Both have the same software specifications. The BeBots drive on a straight way in the same direction. The front-driving BeBot transports a heavy good to the furthestmost place of delivery. The rear-driving BeBot transports several small goods and has to deliver them to several stations. As the front-driving BeBot is heavier than the rear-driving BeBot, its cruising speed is slower than the cruising speed of the rear-driving BeBot. To optimize the energy consumption, BeBots may form a platoon, i.e., the rear-driving BeBot drives in the slipstream of the front-driving BeBot.

During platooning, a collision could occur if the front-driving BeBot must brake very hard (e.g., due to an obstacle on the street) and the rear-driving Be-

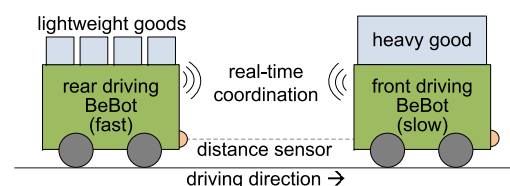


Figure 3: Platoon Scenario with Two BeBots

Bot does not know beforehand that it must brake. To avoid a collision, the front-driving BeBot commands the rear-driving BeBot by sending an asynchronous brake-message to perform a brake maneuver. The brake-message is transmitted to the rear-driving BeBot that is going to brake as soon as it gets this message. This delivery time is safety-critical, because the front-driving BeBot brakes after that time and braking must not result in a collision. A precondition to coordinate such braking behavior is that a BeBot must know if another BeBot is driving behind. Therefore, besides the braking message also messages for starting and ending a platoon are required.

The behavior specification of this scenario can be modeled with statecharts, e.g., to distinguish if a BeBot drives in a platoon or not. By using Dymola, the StateGraph2 library is the first choice. However, the next section shows the limits of StateGraph2 for modeling the behavior of this real-time coordination scenario.

3 Limits of StateGraph2

StateGraph2 [9] is a Modelica library for state-based modeling. It provides the three main classes *Step*, *Transition*, and *Parallel* for modeling statecharts. The class *Step* models discrete system states, the class *Transition* models state changes, and the class *Parallel* models hierarchical and parallel states.

Statecharts are used to describe the behavior of reactive systems. The reactions of such systems are based on their current internal state and the external input. Formalisms for Mealy machines, Harel's statecharts [5], and most common automata-based formalisms support events that can be used for a message-based communication. However, StateGraph2 does not have syntactical constructs. Different steps or transitions can only communicate via shared variables. In real systems, this is not possible when the systems are distributed and have no access to shared memory. The need of shared memory makes it difficult to reuse components as they depend on their environment and not only on their interface description. Therefore, a message-based mechanism is very important. This may be either an asynchronous or a synchronous communication.

StateGraph2 has only a limited support to specify timing behavior. Only the execution of transitions can be delayed. The variable *waitTime* of a *Transition* specifies the time a transition waits before it fires when its guard evaluates to true. If during the waiting period the guard evaluates back to false, the transition does not fire. Therefore, the construct *delayedTransition* of

StateGraph2 can be misinterpreted, because the semantics includes more than a simple delay. In contrast to StateGraph2, Timed automata [1] use clocks to store time independently of a concrete state. Clocks can be read and reset in any state and upon firing of a transition. Therefore, this concept is more flexible for specifying timing behavior. To conclude, the variable *waitTime* alone is too limited to describe real-time behavior.

A modeling language for the software of mechatronic systems that supports hierarchical statecharts as well as synchronous and asynchronous communication, and clocks is MECHATRONICUML [2]. The formal behavior definition of this language is based on timed automata [1]. Therefore, our extensions of the StateGraph2 library are based on concepts of MECHATRONICUML. The next section explains these extensions.

4 Real-Time Coordination Library

As stated above, adequate modeling constructs for synchronous as well as asynchronous communication and for real-time behavior are essential for modern embedded systems. Here, we consider synchronous and asynchronous communication to be a message-based communication where the former means that the sender always waits as long as the receiver is not able to consume the message. The latter means that the sender does not wait on a reaction of the receiver and proceeds with its execution that, in particular, might include sending further messages. For asynchronous communication, this implies that the receiver has to have a message buffer which is sufficiently large to prevent loss of messages.

This section introduces our extended version of the StateGraph2 library, called *real-time coordination library*. In particular, Section 4.1 introduces *synchronization ports* and *synchronization connectors* for synchronous communication. Section 4.2 shows *Messages* and *Mailboxes* for asynchronous communication. Finally, Section 4.3 describes *Clocks*, *Invariants* and *Clock Constraints* for the modeling of real-time behavior according to time automata [1].

4.1 Synchronization Connectors and Ports

For the modeling of synchronous communication, we extended transitions by *synchronization ports* (*sync ports*). Sync ports sub-divide into *sender sync ports* and *receiver sync ports*. A sender sync port of one transition is connected to a receiver sync port of another transition by a synchronization connector. We repre-

sent a sender sync port as a non-filled orange circle, a receiver sync port as a filled orange circle and a synchronization connector as an orange line. In Figure 4, a synchronization connector connects the sender sync port of transition t1 with the receiver sync port t2.

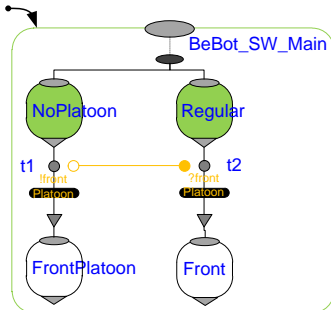


Figure 4: Synchronization Ports and Connectors

A transition that is connected via its sender or receiver sync ports to the receiver or sender sync ports of other transitions is allowed to fire if it is able to fire together with at least one of the connected transitions. For the example in Figure 4, this means that t1 is allowed to fire if t2 is able to fire and vice versa.

We now give a detailed explanation of how the firing of transitions with synchronization is implemented. The implementation is presented with help of the dependency graph in Figure 5.

First, the necessary conditions for firing each of the transitions (without synchronization) have to be satisfied, i.e., the preceding generalized step has to be active, the *condition* of the transition must hold and the optional *condition port* of the transition must be set. If all of these conditions hold, the property *preFire* of each of the transitions will evaluate to true.

Furthermore, if an *after time* is specified for the transition it must have expired. The after time construct is new and replaces the delay (wait) time from the original version of the StateGraph2 library. It differs from the delay time in that *at least* the after time must have expired to let the transition fire. In contrast, the semantics of the delay time is that the delay time must have

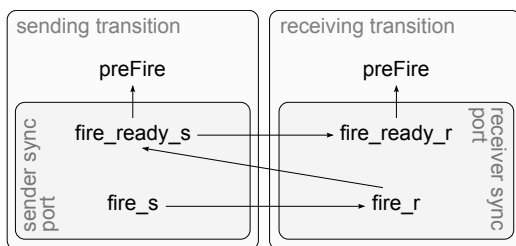


Figure 5: Dependency Graph of Conditions for Firing of Transitions with Synchronization

expired after the transition is fireable in order to let the transition fire. We introduced the after time semantics because it might happen that for two transitions that need to synchronize the time instants in which they are allowed to fire might not match due to their delay time.

If *preFire* of the sending transition, i.e., the transition whose receiver sync port is connected to the synchronization connector, is true, the signal *fire_ready_r* of the receiver sync port is set to true. If for the sending transition, i.e., the transition whose sending sync port is connected to the synchronization connector, holds that *preFire* is true and it receives the signal *fire_ready_r* over its sender sync port then the signal *fire_ready_s* of its sender sync port is set to true. If the signal *fire_ready_s* is true in the receiving transition the signal *fire_r* of the receiver sync port is set to true. Finally, if *fire_r* is recognized to be true in the sending transition the signal *fire_s* of its sender sync port is set to true and both transitions are ready to fire.

4.2 Messages and Mailboxes

For the modeling of asynchronous communication, we introduce two new components named *Message* and *Mailbox*. Each instance of the *Message* component has two purposes. On the one hand, it defines a certain *message type* by specifying an array of formal parameters which might be of type Integer, Boolean or Real. As an example one message type might be defined by the array (*Integer*[2], *Boolean*[1], *Real*[1]). The parameter array of a message type is also called its *signature*. On the other hand, an instance of the *Message* component is responsible for sending a message whenever a connected transition fires. A transition is able to signal to a *Message* component instance to send a message if the *firePort* of the transition is connected to the *conditionPort* of the *Message* component instance.

As a visualization example consider the message type *confirm* in Figure 6. The purple connector connects the *firePort* of the transition t1, displayed as a non-filled purple triangle, to the *conditionPort* of *confirm* where the *conditionPort* of *confirm* is represented by a purple triangle. Additionally, *confirm* has exactly one Integer parameter that is determined by the yellow connector that originates at the port *cruisingSpeed* and connects to the Integer valued input port of *confirm* represented by a yellow filled circle.

For each message type exists exactly one instance of the *Mailbox* component with the same signature. The message type sends its messages to the *Mailbox* instance. To specify which message type belongs to which *Mailbox* instance the *message_output_port* of the

message type is connected to the mailbox_input_port of the Mailbox instance.

A Mailbox instance defines a finite FIFO queue where the size of the queue is settable at design time. In order to let a transition receive a certain message from such a queue its transition_input_port is connected to the mailbox_output_port of the Mailbox instance. Then, the transition is allowed to fire if the Mailbox instance signals that at least one message is present. As an example for the visual representation consider the Mailbox instance confirmBox in Figure 6 that is connected to the transition t2 by a connector.

If two extended StateGraph2 models are included in different component instances they might still communicate asynchronously across the boundaries of these component instances with the help of *delegation ports*. Therefore, one component defines an output delegation port and the other defines an input delegation port. Both delegation ports are connected. Then, the component instance containing the message type connects the message type to the output delegation ports and the component instance containing the Mailbox instance connects the Mailbox instance to the input delegation port. As an example consider Figure 6 which shows two extended State Graph models in two separate component instances communicating over delegation ports that are displayed as envelopes with gray triangle.

Synchronous and asynchronous communication can be combined at one transition. Besides the synchronization conditions the Mailbox instance additionally has to signal to the transition that at least one message is available.

4.3 Clocks, Invariants and Clock Constraints

For the modeling of real-time behavior according to timed automata, we extended the StateGraph2 library

by three components named *Clock*, *Invariant* and *Clock-Constraint*. Clocks are real-valued variables whose values increase continuously and synchronously with time. Clocks might be reset to zero upon activation of a generalized step or firing of a transition. An invariant is an inequation that specifies an upper bound on a clock, e.g., $c < 2$ or $c \leq 2$ where c is a clock. Invariants are assigned to generalized steps and are used to specify a time span in which this generalized step is allowed to be active. A clock constraint might be any kind of inequation specifying a bound on a certain clock, e.g., $c > 2$, $c \geq 5$, $c < 2$, $c \leq 5$ where c is a clock. Clock constraints are assigned to transitions in order to restrict the time span in which a transition is allowed to fire.

As an example consider Figure 7. The example consists of a clock c , an invariant $\text{clockValue} \leq \text{bound}$ and a clock constraint $\text{clockValue} \geq \text{bound}$ where bound is a positive integral number given as a parameter. Clocks are displayed as a rectangle containing a clock icon, invariants are displayed as rectangles containing the corresponding inequation and a transition icon. Clock constraints are displayed as rectangle containing the corresponding inequation and a step icon. The clock which is used by an invariant or a clock constraint is connected via its y port with the clockValue port of the invariants and clock constraints.

When the generalized step *PlatoonProposed* is activated, the clock c is reset to zero, which is accomplished by connecting the activePort (non-filled purple triangle) of *PlatoonProposed* to the u port (non-filled purple circle) of the clock. The invariant is assigned to the step *PlatoonProposed* by the connector originating at the activePort of *PlatoonProposed* leading to the conditionPort (filled purple circle) of the invariant. It means that *PlatoonProposed* is allowed to be active if c has a value less or equal to bound . The transition $t1$ is assigned the clock constraint by connecting the firePort of the clock constraint with the conditionPort of $t1$. The clock

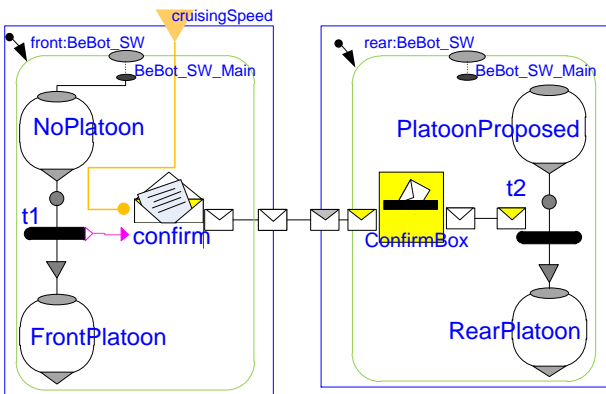


Figure 6: Message Types and Mailbox Instances

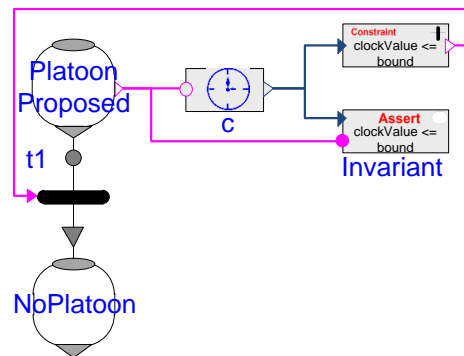


Figure 7: Clocks, Invariants and Clock Constraints

constraint means that t_1 is allowed to fire if c has a value greater or equal to time bound.

5 Formal Definition of the Library

This section covers the formal definition of an extended StateGraph2 model. The Real-Time Coordination library extends the structure of the model given in [9] by synchronization connectors, mailboxes, clocks, invariants and clock constraints whereas the former two are required for synchronous and asynchronous communication resp. and the latter three are used for the specification of real-time behavior analogously to time automata [1]. Due to the possibility of synchronization of two transitions, we altered the delay time of a transition to an after time, which has slightly different semantics.

For the definition of the semantics we give an interpretation algorithm that is analogous to the one given in [9]. Additionally, consider the added elements, i.e., when a generalized step is active the corresponding invariant must not be violated. Further, when a transition fires its clock constraint must be satisfied, it must be able to synchronize, and to receive the required messages.

We present the structure in Section 5.1 and introduce an interpretation algorithm that defines the semantics in Section 5.2.

5.1 Structure

The extension is represented by the following tuple

$$Ext := (Sync, MBox, C, INV, CC)$$

where *Sync* denotes the set of synchronization connectors required for synchronous communication. Let *Msg* be the set of messages used for asynchronous communication. Then $MBox : Msg \rightarrow \mathbb{N}$ determines for each message how often it is available in its corresponding mailbox. The real-time extension is covered by the set *C* of clocks, the set *Inv* of invariants and the set *CC* of clock-constraints.

As said before, the set of messages results from all possible combinations of message parameters. We abstracted from message parameters here, simply saying that there exists a set of distinct messages. Furthermore, in the implementation of our extension there exists the *MailBox* component for the realization of asynchronous communication. Since the number of messages included in a certain mailbox suffices to be able to determine whether a transition that requires such a message is able to fire, we abstracted from the

mailboxes here in form of the *MBox* function. The following definition consists of elements that were already defined in [9]. For the sake of completeness, we describe and list them.

With the help of our extension *Ext*, we define an *extended StateGraph2 model (ESGM)* Γ as follows:

$$\Gamma := (V_c, G, T, G_I, G_E, Ext)$$

where

- V_c is a set of Boolean expression as defined in [9].
- G is a set of generalized steps $G = \{g_1, g_2, \dots\}$
A generalized step g_i is defined as a 7-Tuple

$$g_i = (\Gamma_s, I, O, S, R, Inv_{g_i}, RESET_{g_i})$$

where

- Γ_s is a possibly empty set of sub-graphs $\Gamma_s = \{\gamma_1, \gamma_2, \dots\}$. A sub-graph $\gamma_i \in \Gamma_s$ is again an ESGM. Note that this recursive definition allows an arbitrary deep nesting of ESGMs.
 - I is a vector of in (entry) ports $I = [i_1, i_2, \dots]$. An in port is a connection point incoming transitions of g_i are connected to.
 - O is a vector of out (exit) ports $O = [o_1, o_2, \dots]$. An out port is a connection point outgoing transitions of g_i are connected to.
 - S is a possibly empty vector of suspend ports $S = [s_1, s_2, \dots]$. A suspend port is a connection point outgoing transitions of g_i are connected to. The difference to out ports is that the active generalized steps of sub-graphs of g_i are stored for later restore.
 - R is a possibly empty vector of resume ports $R = [r_1, r_2, \dots]$. A resume port is a connection point ingoing transitions of g_i are connected to. The difference to in ports is that the active generalized steps of sub-graphs of g_i that were active when g_i was left by a suspend port are restored.
 - $Inv_{g_i} \subseteq Inv$ is a set of invariants. An invariant describes that a clock must never exceed a certain bound when the generalized step is active. It is denoted as an inequation of the form $c \leq n$, where $c \in C$ is a clock and $n \in \mathbb{N}$ is a natural number (including zero).
 - $RESET_{g_i} \in C$ is a set of clocks that are to be reset to zero when the generalized step is activated.
- A generalized step that has in and out ports but no other ports and no sub-graphs, i.e., $I \neq \emptyset$, $O \neq \emptyset$ and $R = S = \Gamma_s = \emptyset$ is called *step*. A generalized step that has resume ports, suspend ports or sub-graphs, i.e., $R \neq \emptyset$, $S \neq \emptyset$ or $\Gamma_s \neq \emptyset$ holds, is called *parallel step*.
- T is a set of transitions $T = \{t_1, t_2, \dots\}$. A transition

$t_i \in T$ is defined by the 10-tuple

$$t_i = (p_{t_i}^{IR}, p_{t_i}^{OS}, C_{t_i}, A_{t_i}, CC_{t_i}, R_{t_i}, S_{t_i}^R, S_{t_i}^S, M_{t_i}^R, M_{t_i}^S)$$

where

- $p_{t_i}^{IR}$ is a connected port of an in or resume vector of a succeeding generalized step $g_i \in G$.
- $p_{t_i}^{OS}$ is a connected port of an out or suspend vector of a preceding generalized step $g_i \in G$.
- $C_{t_i} \in V_c$ is the fire condition associated with t_i .
- $A_{t_i} \in \mathbb{R}$ is the after time associated with t_i . Note, that we consciously chose the name after time instead of delay time as in the original definition in [9] since the semantics of the after time will be different from the one of the delay time.
- $CC_{t_i} \in CC$ are the clock constraints associated with t_i .
- $R_{t_i} \in C$ are the clocks to be reset when t_i fires.
- $M_{t_i}^R \subseteq Msg$ is the message that must be received when t_i fires.
- $M_{t_i}^S \subseteq Msg$ is the message that is sent when t_i fires.
- $S_{t_i}^R \subseteq Sync$ is the synchronization connector that has to be set by another transition when t_i fires.
- $S_{t_i}^S \subseteq Sync$ is the synchronization connector that is set if t_i is fireable.

We further define that a transition might have at most one message that is to be received and at most one message that is to be sent, i.e., $|M_{t_i}^R| \leq 1$ and $|M_{t_i}^S| \leq 1$ resp., and at most one synchronization connector over which a signal is sent or received, i.e., $|S_{t_i}^R| + |S_{t_i}^S| \leq 1$.

- $G_I \subseteq G$ contains the initial generalized step of Γ .
- $G_E \subseteq G$ contains the exit generalized step of Γ .

As a well-formedness constraint, we assume that every ESGM has exactly one initial state and at most one exit state, i.e., $|G_I| = 1$ and $|G_E| \leq 1$. Furthermore, we assume that the *uppermost ESGM* $\Gamma = (V_c, G, T, G_I, G_E, Ext)$, i.e., that ESGM that is not embedded by any other ESGM, does not have an exit generalized step, i.e., $G_E = \emptyset$.

5.2 Interpretation Algorithm

1. Activate the initial generalized step $g \in G_I$. If g has sub-graphs, then recursively activate the initial generalized steps of all of its embedded sub-graphs.
2. Determine the set $T_{fireable}$ of all transitions t_i that satisfy:
 - its condition C_{t_i} is true,
 - the required after time A_{t_i} has passed,
 - its in or resume port $p_{t_i}^{IR}$ is set to true,
 - if its preceding generalized step has sub-graphs, the exit generalized steps of all of these sub-

graphs are recursively activated

- if $M_{t_i}^R \neq \emptyset$ and $m \in M_{t_i}^R$ is the message to be received by t_i , the Mailbox of m contains at least one message, i.e., $MBox(m) > 0$.
 - there exists no other transition $t_j \in T_{fireable}$ that has the same preceding generalized basic step and has higher priority than t_i where the priority results from the index of the transition in the port vector (see [9]).
3. For all $t_i \in T_{fireable}$ do:
 - i. if $S_{t_i}^S \neq \emptyset$ and $s \in S_{t_i}^S$ is the synchronization connector of t_i for sending a signal, set s to true
 4. Determine the set $T_{syncable}$ of all transitions $t_i \in T_{fireable}$ that satisfy:
 - either $S_{t_i}^R = \emptyset$ or
 - if $S_{t_i}^R \neq \emptyset$ and $s \in S_{t_i}^R$ is the synchronization connector of t_i , t_i is set to true
 5. For all $t_i \in T_{syncable}$ fire t_i as follows:
 - i. Deactivate the preceding generalized step g of t_i . If g_i includes sub-graphs deactivate these sub-graphs recursively.
 - ii. Activate the succeeding generalized step g' of t_i . If g' includes sub-graphs activate these sub-graphs recursively as follows:
 - if t_i is connected to g' by a resume port, the generalized steps of g' and of all sub-graphs of g' that were active the last time g' was active are recursively activated
 - else, activate all initial generalized steps of g' and its sub-graphs recursively.
 - iii. if $M_{t_i}^R \neq \emptyset$ and $m \in M_{t_i}^R$ is the message to be received by t_i , then take one message out of the the Mailbox of m , i.e., $MBox := (MBox \setminus \{(m, d)\}) \cup \{(m, d-1)\}$ where $d \in \mathbb{N}$ is the amount of messages in the mailbox before t_i fires.
 - iv. if $M_{t_i}^S \neq \emptyset$ and $m \in M_{t_i}^S$ is the message to be sent by t_i , then put one message into the Mailbox of m , i.e., $MBox := (MBox \setminus \{(m, d)\}) \cup \{(m, d+1)\}$ where $d \in \mathbb{N}$ is the amount of messages in the mailbox before t_i fires.
 6. Goto 2.

6 Case Study

This section shows how we modeled the platoon scenario. First, we used the StateGraph2 library in combination with our Real-Time Coordination library to specify the discrete software. Then, we connected the software model with the simulation model of the BeBot hardware that we have presented in Section 2.1. This section shows an excerpt of our model. The complete

model is delivered within our Real-Time Coordination library.

Figure 8 shows the discrete behavior specification that we modeled as class *BeBot_SW* in *Dymola*. We used the *Step* components and the *Parallel* component from the *StateGraph2* library. From the Real-Time Coordination library, we used the *Transition* components, the *Message* components, the *Mailbox* components, and the *DelegationPort* components. We omit guards, connection lines between synchronizations, and timing constraints.

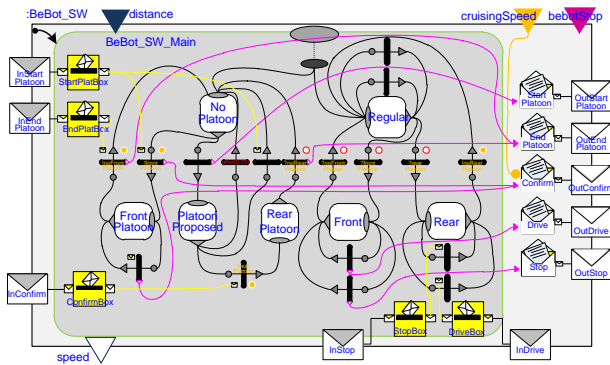


Figure 8: Platoon Scenario Behavior Modeled

The interface of the class *BeBot_SW* defines three incoming parameters: the distance to a BeBot, that drives in front, the *cruisingSpeed* of the BeBot, and *bebotStop* that defines if the BeBot has to stop. The outgoing parameter is the speed of the BeBot. Furthermore, five asynchronous messages are defined that can be sent and received: *StartPlatoon* to propose to start a platoon, *Confirm* to confirm the start proposal, *EndPlatoon* to command the end of the platoon, *Stop* to command a rear-driving BeBot to stop, and *Drive* to inform a rear-driving BeBot that it no longer has to stop.

Within *BeBot_SW*, two parallel branches were defined. The first branch handles the platoon activation and deactivation and consists of the steps *NoPlatoon*, *PlatoonProposed*, and *FrontPlatoon*. The second branch handles the coordinated braking within a platoon and consists of the steps *Regular* (a BeBot has no limitations regarding braking), *Front* (a BeBot has first to inform the rear-driving BeBot before braking), and *Rear* (a BeBot must brake when the front-driving BeBot commands it). The synchronization between the two branches is realized by using synchronous communication, e.g., if step *FrontPlatoon* is activated, then step *Front* will also be activated at the same time. Among others, this class contains a timing constraint that the state *PlatoonProposed* is no longer active than 50ms.

Figure 9 shows the two connected instances front and

rear of the class *BeBot_SW*. Furthermore, it shows two instances of the *BeBot* hardware model (see Figure 2) and how they are connected with the software models. The instance *distance* of the class *Distance* calculates the distance of the rear BeBot to the front BeBot. We do not display the connections to the inputs *cruisingSpeed* and *stop*.

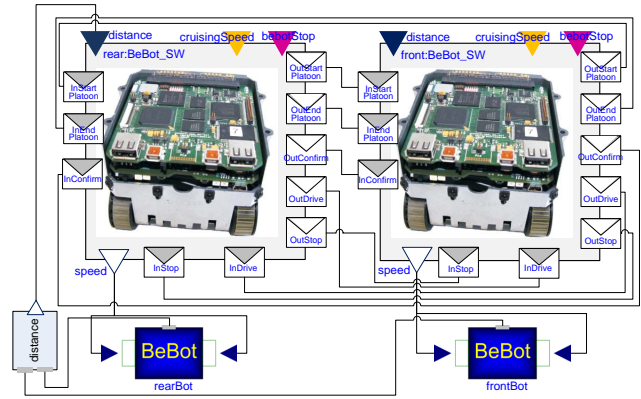


Figure 9: Platoon Scenario Instance Model

Figures 10 and 11 show the results of a simulation run of the model. Figure 10 shows the asynchronous messages that were sent between the rear- and the front-driving BeBot. Figure 11 shows the speed result of both BeBots during a performed simulation. Right at the start, the rear-driving BeBot speed was higher. As the distance had reached a size where a platoon was needed, the rear-driving BeBot sent the message *StartPlatoon*. At time 8.6, the rear-driving BeBot received the message *Confirm(1)* so it had adjusted its speed to 1. At time 25, the stop input of the rear-driving BeBot raised to 1. Therefore, the rear-driving BeBot ended the platoon by sending the message *EndPlatoon* and stopped for 10s. Then the rear-driving BeBot started again to close the gap by driving faster and to start a new platoon.

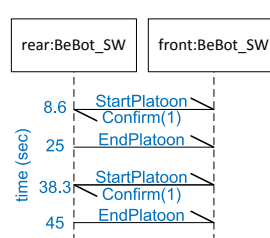


Figure 10: Sequence Diagram

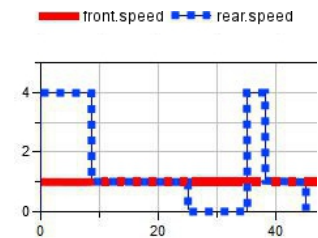


Figure 11: Simulation Plot

Figure 12 shows the 3D view of the simulation run. The left shows the moment when the rear BeBot drives faster than the front BeBot and the right shows when both BeBots drive in the platoon with the same speed.

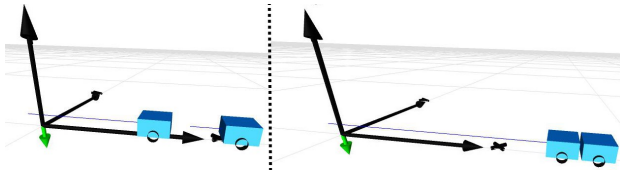


Figure 12: 3D View of Simulation

7 Related Work

This section presents some other approaches for modeling discrete state-based behavior for simulating hybrid cyber-physical systems. We focus on the capabilities to model and simulate real-time properties and constraints of the behavior, synchronize parallel behavior, and to communicate via asynchronous messages.

7.1 SimulationX

SimulationX supports an own representation of state machines which follows the model of UML state machines but only implements a limited subset [3]. SimulationX state machines have no support for parallel behavior and therefore no support of synchronizations. The asynchronous signals have no support for an arbitrary number of parameters and are lost when the receiver is not enabled to consume them immediately. They have no concept of a mailbox for storing messages. SimulationX supports only limited timing support. Its time events only react to an expression which is relative to the active state time of the transition which is triggered by the *after* event. It is not possible to model time invariants as first class entities. As SimulationX supports Modelica, it is possible to port the concepts that we present in this paper to SimulationX.

7.2 ModelicaML

ModelicaML is a UML Profile [13] which extends UML *Classes* and *Properties* with *Stereotypes* for Modelica. Therefore, it is possible to model with UML Classes as in Modelica. Further, ModelicaML defines a mapping of UML state machines and simple internal events to plain Modelica algorithmic code [14]. For more complex messages it is possible to use external C-functions [11]. A code generation algorithm does the mapping of UML state machines to Modelica code automatically. In contrast to the State Graph2 extension presented in this paper it is hard to edit the state machine behavior directly in Modelica because it is encoded in a complex algorithm. Further, ModelicaML has no support for synchronization of parallel behavior

from different regions as presented in Section 4.1. ModelicaML supports only rudimentary timing behavior as first class entity with its AFTER-macro [14]. This construct is a transition guard relative to the active time of a state. ModelicaML also does not support time invariants of states. As ModelicaML supports Modelica, it is possible to port the concepts that we present in this paper to ModelicaML.

7.3 MATLAB/Simulink, Stateflow

MATLAB provides the custom modeling language Stateflow for state based behavior. Stateflow has interfaces to the Simulink environment. Stateflow has some drawbacks for modeling communication protocols with real-time requirements between distributed systems. For clocks, helping elements from Simulink to count time-ticks are needed. Stateflow also has no concept of asynchronous, message-based communication with mailboxes for sent and received messages. Stateflow events are not buffered by the receiver and could be lost if the receiver is busy. It is possible to encode asynchronous message-based communication. Therefore, you need a complex combination of several linked Simulink and Stateflow blocks, which is hard to maintain manually [6, 10].

8 Conclusions and Future Work

Today, autonomous embedded systems are increasingly connected to each other to realize new innovative functionality, e.g., in the case of vehicle-to-vehicle communication to realize platooning.

We presented an extension of the StateGraph2 library that enables modeling a real-time communication and coordination between autonomous embedded systems by providing library elements for asynchronous and synchronous communication as well as real-time constraints. We modeled two miniature robots that drive in a platoon with our library to simulate it.

We plan to make several additions to our library. Asynchronous message exchange between autonomous systems may suffer from message loss or message delays. Therefore, we plan to enable modeling different probabilistic quality of service characteristics, e.g., message delays and message losses. The new Modelica version 3.3 have built-in support of finite state machines, which makes the StateGraph2 library obsolete. However, the new built-in finite state machines does not support asynchronous message-based communication, so we suggest to use our extensions for asynchronous

message-based communication. The integration is up to further research.

With respect to tool chains, we want to implement automatic transformations from MECHATRONICUML to the presented extended StateGraph2 library. This allows us to reap the benefits from formal verification by model checking, which is possible for models of the MECHATRONICUML [2], and integrated simulation including feedback controllers and physics by using Modelica. Finally, we will use our library in several other case studies, including a de-centralized industrial dough mixing system.

Acknowledgments

This work was developed in the project 'ENTIME: Entwurfstechnik Intelligente Mechatronik' (Design Methods for Intelligent Mechatronic Systems). The project ENTIME is funded by the state of North Rhine-Westphalia (NRW), Germany and the EUROPEAN UNION, European Regional Development Fund, 'Investing in your future'. Chia Choon Loh is supported by the International Graduate School Dynamic Intelligent Systems.

References

- [1] R. Alur and D.L. Dill. A theory of timed automata. *Theoretical computer science*, 126(2):183–235, 1994.
- [2] S. Becker, C. Brenner, S. Dziwok, T. Gewering, C. Heinzemann, U. Pohlmann, C. Priesterjahn, W. Schäfer, J. Suck, O. Sudmann, and M. Tichy. The mechatronicuml method - process, syntax, and semantics. Technical Report tr-ri-12-318, Software Engineering Group, Heinz Nixdorf Institute, University of Paderborn, 2012.
- [3] U. Donath, J. Haufe, T. Blochwitz, and T. Neidhold. A new approach for modeling and verification of discrete control components within a Modelica environment. In *Proceedings of the 6th Modelica Conference, Bielefeld*, pages 269–276, 2008.
- [4] C. Ebert and C. Jones. Embedded software: Facts, figures, and future. *IEEE Computer*, 42(4):42–52, 2009.
- [5] D. Harel. Statecharts: A visual formalism for complex systems. *Science of computer programming*, 8(3):231–274, 1987.
- [6] C. Heinzemann, U. Pohlmann, J. Rieke, W. Schäfer, O. Sudmann, and M. Tichy. Generating simulink and stateflow models from software specifications. In *Proceedings of the International Design Conference, DESIGN 2012, Dubrovnik, Croatia*, May 2012.
- [7] S. Herbrechtsmeier, U. Witkowski, and U. Rückert. Bebot: A modular mobile miniature robot platform supporting hardware reconfiguration and multi-standard communication. In *Progress in Robotics, Communications in Computer and Information Science. Proceedings of the FIRA RoboWorld Congress 2009*, volume 44, pages 346–356, Incheon, Korea, 2009. Springer.
- [8] C. C. Loh and A. Trächtler. Laser-sintered platform with optical sensor for a mobile robot used in cooperative load transport. In *Proceedings of the 37th Annual Conference on IEEE Industrial Electronics Society*, pages 888–893, November 2011.
- [9] M. Otter, M. Malmheden, H. Elmqvist, S.E. Mattsson, C. Johnsson, D. Systèmes, and S.D. Lund. A new formalism for modeling of reactive and hybrid systems. In *Proceedings of the 7th Modelica'2009 Conference, Como, Italy*, 2009.
- [10] M. Pajic, Z. Jiang, I. Lee, O. Sokolsky, and R. Mangharam. From verification to implementation: A model translation tool and a pacemaker case study. In *Proceedings of the 18th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2012), Beijing, China*, April 2012.
- [11] U. Pohlmann and M. Tichy. Modelica code generation from ModelicaML state machines extended by asynchronous communication. In *Proceedings of the 4th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools, EOOLT 2011, Zurich, Switzerland*, 2011.
- [12] W. Schäfer and H. Wehrheim. The Challenges of Building Advanced Mechatronic Systems. In Lionel C. Briand and Alexander L. Wolf, editors, *FOSE*, pages 72–84, 2007.
- [13] W. Schamai. Modelica modeling language (ModelicaML): A UML profile for Modelica. Technical report, Linköping University, Department of Computer and Information Science, The Institute of Technology, 2009.
- [14] W. Schamai, U. Pohlmann, P. Fritzson, C. J.J. Paredis, P. Helle, and C. Strobel. Execution of uml state machines using modelica. In *Proceedings of EOOLT*, pages 1–10, 2010.
- [15] C. Weiß. V2X communication in Europe - From research projects towards standardization and field testing of vehicle communication technology. *Computer Networks*, 55(14):3103–3119, 2011.

Implementation of a Graphical Modelica Editor with Preserved Source Code Formatting

Tobias A. Mattsson^a Jon Sten^a Tove Bergdahl^c Jesper Mattsson^c Johan Åkesson^{b,c}

^aDepartment of Computer Science, Lund University, Sweden

^bDepartment of Automatic Control, Lund University, Sweden

^cModelon AB, Sweden

Abstract

When an Integrated Development Environment (IDE) is developed, the support for multiple views of the same document is often essential. An example of this is Modelica models, where it should be possible to view and edit the same model in both its textual and graphical representation.

One implementation of Modelica is the open source platform JModelica.org. It contains the Eclipse-based JModelica.org IDE, providing a text editor for Modelica code based on the Eclipse platform.

In this paper, we present an implementation of a graphical editor for the JModelica.org IDE. Several challenges arising when implementing a graphical editor for Modelica models are discussed. Amongst others, the difficulties in rendering Modelica diagrams and how to interact with existing frameworks in Eclipse are covered. Also, a method for preserving the formatting of a modified source code file is presented, which is essential when the model is altered in the graphical editor.

The presented implementation is compared to other open source software (OSS) implementations of Modelica editors.

Keywords: AST; JModelica.org; Eclipse; GEF; Graphical Editing; Icon Rendering; Preserved File Formatting; Pretty Printing

1 Introduction

Simulation and optimization of dynamic systems is becoming a standard tool in several industrial branches. The trend is mainly driven by the demand for decreased product time to market and shortening the development time, by substituting system prototyping for simulation. Modelica is one of many domain spe-

cific languages developed with the goal to meet the demand of such model-based design languages.

One implementation of the Modelica language is the JModelica.org platform [1]. It contains a Modelica compiler as well as an Integrated Development Environment (IDE) for Modelica code. Currently, a comprehensive text editor for editing Modelica source code is available in the JModelica.org IDE [2], allowing the developer to define new models based on equations and existing models. The JModelica.org IDE is implemented using the Eclipse framework¹ which is a modular, extensible application framework for IDEs.

In this paper, we present an implementation of a graphical editor for Modelica in the JModelica.org platform which will complement the textual editor, already available in the JModelica.org IDE. The editor is implemented as an Eclipse plugin using the Graphical Editing Framework (GEF)² which is a framework for creating graphical editors, developed for the Eclipse platform. The graphical editor communicates and modifies Modelica models through an abstract syntax tree (AST). It also features preserved file formatting in the JModelica.org IDE. The work presented in this paper is the result of two master's theses [3, 4], conducted at Modelon AB.

This paper is outlined as follows. In Section 2, a brief background of the JModelica.org platform is given and the compiler construction framework JastAdd [5] is introduced. The Eclipse project and the Graphical Editing Framework (GEF) are also introduced in this section. In Section 3, a comparison to similar OSS tools is presented. The implementation of the graphical editor is discussed in Section 4 and Section 5 summarizes this paper.

¹<http://eclipse.org>

²<http://eclipse.org/gef>

2 Background

2.1 JModelica.org

JModelica.org is an open source project for optimization and simulation of complex dynamic systems. The JModelica.org platform includes compilers for Modelica and the Modelica language extension Optimica [6], as well as an integration to the simulation package Assimulo [7]. An interface to the compilers and simulation and optimization algorithms is available in Python, which enables scripting of the typical modeling and optimization activities.

Also part of the JModelica.org platform is an IDE for Modelica. The JModelica.org IDE is implemented as a plugin in Eclipse using the JModelica.org compilers and the JastAdd framework. The IDE provides textual editing support such as syntax highlighting, code folding, code outline, brace matching and error checking of models.

2.2 Eclipse

The Eclipse Foundation is an open source community whose aim is to produce open development platforms with comprehensive extension frameworks³. The IDE is heavily modularized so that it is possible to add, remove and extend functionality with a small amount of code and without altering any core source files. The modularization also makes it possible to create different bundles, including different editors and views. For instance, there is the Eclipse Software Development Kit (SDK) that includes a comprehensive Java Development Tool (JDT) for Java development and also the C++ Development Tool that is an IDE for C and C++. These are two different development environments with different functionality, yet they still use the same base IDE and base functionality.

2.3 GEF

The Graphical Editing Framework (GEF) is one of the most popular frameworks for graphical editing in Eclipse and it is also the one used for the editor in this paper⁴. GEF is a rather complicated system with many design patterns and classes. When developing graphical editors using GEF, the developer has to define two types of classes, *EditParts* and *EditPolicies*.

EditParts is the most basic part of GEF. These classes join the document model with the view. There

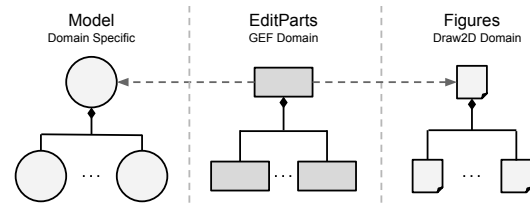


Figure 1: View of the document model, EditPart and figure tree and their linkage.

is usually a one to one representation between document model nodes and EditPart classes. The view is represented by figures. Normally, figures also map one to one with EditParts, see Figure 1.

EditPolicies handle the interaction with the user, the Eclipse framework and the underlying model. For example, the graphical editor specifies an *EditPolicy* that determines what should happen when the user tries to move a component. If the move is valid, it will create a move command that alters the model component definition.

An *EditPolicy* specification usually only handles a single task or a group of related tasks. Using this pattern means that the user interaction is separated from the *EditParts*. Instead, the interaction is handled by *EditPolicies* installed on *EditParts*. This enables different *EditParts* with similar behavior to use the same *EditPolicies*, which reduces code complexity. It is also convenient for the developer, since it allows for gradually extending the functionality with new features.

2.4 JastAdd

JastAdd⁵ is an open source meta-compilation system that is used for compiler generation and other programs that have the need to analyze code. It provides means to define attribute grammars [8], and introduces the possibility to use *aspect-oriented programming* (AOP) when constructing a compiler. With aspects, the source files describe a certain behavior or functionality, rather than objects in object-oriented programming (OOP). In other words, the behavior for several different objects may be defined in the same aspect.

JastAdd code is organized in abstract grammar files and aspect files. These source files are collected and the functionality from the aspects is woven into Java files before they are finally compiled.

The JastAdd project provides a framework for supporting IDEs based on Eclipse [2]. It consists of a generic IDE plugin with supporting classes and default

³<http://eclipse.org>

⁴<http://eclipse.org/gef>

⁵<http://jastadd.org>

aspects for attributes. The attributes provide common services such as code folding and code outline. The main parts of the generic IDE plugin are the *builder* and the *registry*. When the Eclipse framework needs a build, it triggers the builder. The builder then delegates the work to a compiler. When the compiler is done, it provides an AST for files or projects. These ASTs are cached in the registry.

2.5 Graphical Annotations

In this paper, Modelica annotations, or more specifically graphical annotations, will play an important part since they are used for representing a model and its components graphically. A graphical editor uses the information in the graphical annotations when rendering icons and diagram. The editor also modifies annotations when the user makes changes in the graphical editor.

Listing 1: Code example of graphical annotations in the three different locations permitted.

```

model LowPass
  ...
  Analog.Basic.Resistor R1
  annotation (Placement( transformation (
    extent={{-25, -25},{25, 25}},
    origin={-25, 50}
  )));
  ...
equation
  ...
  connect (R1.n, p2)
  annotation (Line(points = ...));
  ...
  annotation (
    Icon (
      coordinateSystem (extent = ...),
      graphics = {...}
    )
  );
end LowPass;

```

There are three locations where graphical annotations may appear:

- (a) Directly after a component definition. If specified, it will define how that component should be rendered, size and origin.
- (b) Directly after a connect statement. If specified, it will define how the connection should be rendered, line points and its color.
- (c) At the end of a model definition. If specified, it will define how the icon and diagram of the model should be drawn.

Examples of the three different locations are illustrated in Listing 1.

3 Related Work

There are several approaches to formatting preservation and graphical Modelica editing. In this section, a comparison will be given to some of the popular open source alternatives.

3.1 Formatting Preservation

OpenModelica⁶ is another open source initiative based on Modelica. The OpenModelica environment also has procedures for preserving formatting. Peter Fritzson et al. describes these procedures as an initiative to preserve comments and indentation when refactoring Modelica code [9].

OpenModelica stores the text representation of the code in a separate tree, which has the same structure as the original AST. In this way, they are able to avoid cluttering the AST with text positions. The text representation is created piece-wise when needed. The nodes in this separate tree stores the text positions and have a one-to-one mapping with the nodes in the AST.

While the solution for preserving formatting presented in this paper also aims to do most of the work when it is actually needed, it does more work in the parsing process. The position of the nodes and the actual formatting text and type is extracted during parsing. This data is then associated to the source AST. This makes the AST slightly more memory consuming at first, compared to the OpenModelica solution.

The most significant difference between the two solutions is that in JModelica.org, the formatting always resides in the source AST while OpenModelica stores it in a separate tree. Although this makes the source AST in JModelica.org more verbose, the advantage is that modifying the source AST does not require any synchronization with a second tree.

Maartje de Jonge and Eelco Visser have also presented an algorithm for preserving the original layout of source code when modifying an AST [10]. Their algorithm relies on text reconstruction and origin tracking. The algorithm stores a reference to the leftmost and rightmost token in the stream for each node in the AST, which in turn holds the corresponding start and end offset. When the AST is to be modified, the nodes and their positions in the new tree are traced back to

⁶<http://openmodelica.org>

their origin. The text can then be reconstructed from this origin.

The algorithm also comes with an intelligent heuristic for associating comments with the correct node. Cases such as block comments, comments before and after a line of code, inside comma separated lists, code removed by commenting and multi-line comments beside multiple statements are discussed. Suggestions how to handle most of these cases are also described.

The solution presented in this paper is less involved than the algorithm by de Jonge and Visser, but still covers the realistic cases threatened in this paper. All comments in the source code that are located on the right-hand side of a node, but before a line break or the next node, are associated with that node. Any comments that follow are considered to belong to the next node, and so forth. It is important to remember that comments are meant for people, not machines, to read and interpret. Thus, there are no predefined rules for how to relate comments to code and it is practically impossible to perfect such an algorithm.

As the JModelica.org IDE is an Eclipse plugin it is worth mentioning how the Eclipse Java Development Tools (JDT) handles changing the AST. The Eclipse JDT has an API for refactoring code using the AST [11]. When AST nodes are added, removed or replaced in JDT, these operations are translated into text edits which can then be applied to the original source. This is a very different approach than ours, as this means that the original AST is never touched by JDT. Instead the source text is edited and the AST is then updated from that source. In JModelica.org, such an approach would require a total recompilation of the code with every AST modification as there is currently no way to incrementally compile the source code. This would thus not be an adequate solution, as it would most likely make the editor very slow.

3.2 Graphical Modelica Editors

OMEdit is an editing front-end to the OpenModelica compiler. It contains tools for model creation, diagram editing, icon editing, simulation, plotting, documentation view and text editing mode. The editor was developed as part of a thesis by Asghar, Syed Adeel and Tariq, Sonia in 2010 [12, 13].

OMEdit uses the CORBA interface to communicate with OpenModelica. CORBA is supplied by OpenModelica and allows for interaction between an application and its AST. OMEEdit uses it to retrieve and store information such as: model structure, annotation, documentation, simulation and graph plotting.

OMEdit is mainly developed in C++ and relies on *QT*⁷ for graphical UI handling and rendering of graphical primitives. QT is a cross platform framework that allows the developer to rapidly develop Graphical User Interface (GUI) based applications that work on multiple platforms. It also has support for multiple target languages like C++ and JavaScript.

Compared to the graphical editor presented in this paper, OMEEdit is more comprehensive. It defines a complete IDE with text editor, parameter view, simulation view and graphical editor.

There are some significant differences between the two graphical editors, besides the programming language. OMEEdit uses QT to generate a GUI, as opposed to GEF that is used for the graphical editor in this paper. GEF and QT provide the same basic functionality but for different programming languages and platforms. However, QT has more extensive support for the graphical features that are specified in Modelica than GEF. QT also takes care of all transformation and rendering of graphical primitives.

Performance wise there are some small differences between the two graphical editors. When adding and removing components a noticeable lag is present in OMEEdit editor while it happens instantly in JModelica.org editor. This is most likely a result of the CORBA interface and the fact that OMEEdit does not operate directly on the AST.

OMEdit also lacks some basic features like an undo and redo stack. This is likely due to that support for this is missing in QT. This feature is something that was supported by GEF and is one of the essential features in the graphical editor described in this paper.

SimForge⁸ is another open source toolkit that is based on OpenModelica. It offers similar features as the JModelica.org IDE and OMEEdit. It has a text editor as well as a graphical editor that allows for both diagram and icon editing. Additionally it has a parameter editor that allows for modification of component values. The SimForge project has been inactive for some time and there is also a lack of information about the implementation and the frameworks used. Therefore, no thorough comparison is given in this paper.

⁷<http://qt.nokia.com>

⁸<http://trac.ws.dei.polimi.it/simforge/>

4 Implementation

4.1 Compiler Architecture

The JModelica.org compiler is divided into two parts, front-end and back-end. The front-end is responsible for parsing, AST building, error checking and flattening of Modelica models. The front-end builds three ASTs, the source AST, the instance AST and the flat AST.

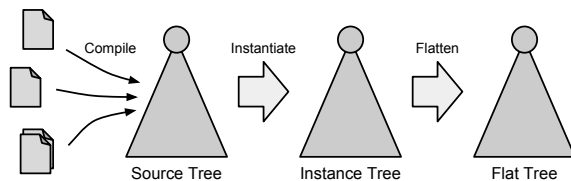


Figure 2: The three ASTs and the different compilation stages.

Source AST is the first AST that is built. It is almost a direct translation of the Modelica model into a tree. The source AST is necessary when calculating the instance tree.

Instance AST is instantiated from the source AST. All component declarations has been resolved and expanded with the contents of their classes.

Flat AST is the flattened version of the model. It is reduced from the instance tree and consists of a list of equations and variables.

A brief overview of the three ASTs can be seen in Figure 2.

4.2 Graphical Editor

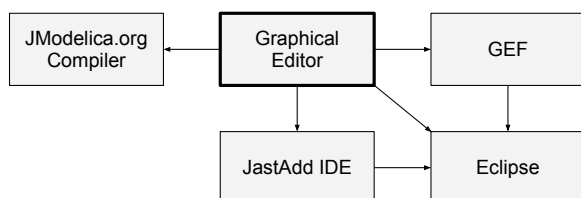


Figure 4: An overview of the interaction between the graphical editor and the other components.

An overview of the design and the different components that the graphical editor relies on is shown in Figure 4. The graphical editor communicates with the JModelica.org compiler to retrieve graphical annotations as well as Modelica class structure. All editing is saved back into the source and instance AST and the

icon structure. The icon structure is an abstraction of the structure that is used when representing Modelica annotations in the source AST. It is built to resemble the structure of a graphical annotation as it is defined in the Modelica specification. GEF is an obvious component that the editor relies heavily on. GEF helps the editor with synchronization between the EditParts and icon structure in the compiler. It is also responsible for interaction with the Eclipse framework and low level rendering of graphics in the editor. The graphical editor also has some direct interaction with Eclipse, such as the ability to open a component for modification of parameters on sub components. The JastAdd IDE framework is used as an interface to the JModelica.org compiler when compiling classes.

4.2.1 EditParts

When constructing a GEF editor, it is important to make a good design between EditParts and the underlying document model. Normally, there is one EditPart class for each document model node. The graphical editor presented in this paper is no exception and uses one EditPart class for each icon structure node. For example, the icon structure node *Rectangle* is represented by a *RectangleEditPart*. The *RectangleEditPart* handles all the rendering and interaction with the graphical user interface (GUI). By using EditPolicies, it is possible to alter the behavior and control what happens when the rectangle is moved or resized. Similarly, there will be one EditPart class for each node type in the icon structure, specifying the behavior of that node.

4.2.2 Rendering

Once an EditPart has been produced from the icon structure, it creates an appropriate figure and populates it with the correct attributes from the icon structure. For example, the rectangle mentioned in Section 4.2.1 will populate its figure with width, height and rotation. It will also set line color, line pattern and fill color.

It can sometimes be troublesome to render Modelica models in GEF. Modelica supports both rotation and scaling of graphics whilst GEF does not support rotation and has limited support for scaling. In the graphical editor this is solved by transforming the points that the graphical object consists of. The transformation is done using Euclidean transformations, that are built hierarchically over the component structure of the Modelica model.

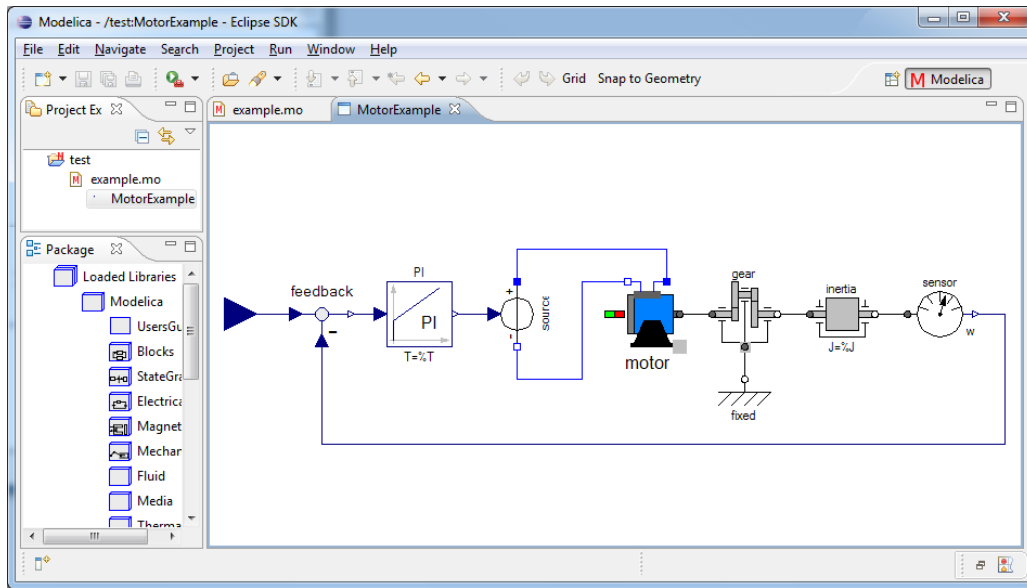


Figure 3: Screen shot of the graphical editor.

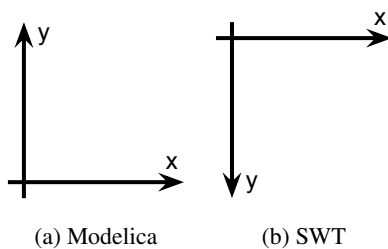


Figure 5: Difference between Modelica and GEF coordinate system with positive direction indicated by arrows on the axis.

Finally, when rendering models it is important to convert between the handedness of the coordinate systems used by Modelica and GEF respectively. Modelica uses a right-hand coordinate system, see Figure 5a while GEF uses left-hand coordinate system, see Figure 5b. If no consideration is taken to handedness the resulting image will be upside down. The solution is simple, once all transformations are done the image is flipped along the y-axis.

4.3 AST Communication

Changes made by the user in the graphical editor has to be propagated back into the icon structure and underlying source AST. These propagations can be divided into two categories, annotation editing and structural editing.

4.3.1 Annotation Editing

Some graphical changes, such as moving or resizing components, are localized and only affects annotations in the code. Most of the operations performed while graphically editing a model falls into this category. Since this kind of change only affects the source AST, it is simple to perform.

4.3.2 Structural Editing

Structural editing is a more complicated type of editing operation which is performed on the AST. It occurs when a *component* or a *connection* is added to or removed from the model. The main challenge is that both the source and instance tree must be updated consistently. In the current implementation, the component or connection is first added to the source tree. The instance tree then instantiates a new component or connection from the source node, resulting in a consistent result for the currently opened model, see Figure 6. Removing a component is performed in the reverse order as opposed to adding. First, the component or connection is removed from the instance tree and then in the source tree.

There are, however, side effects that are not handled. If the edited model is used as a component in another model and that other model is also open, the latest changes will not appear until that model is reloaded. A possible solution is to sense when a structural edit has occurred and in that case reload the editor. Another solution is to propagate any changes in a model to all instances of the same model.

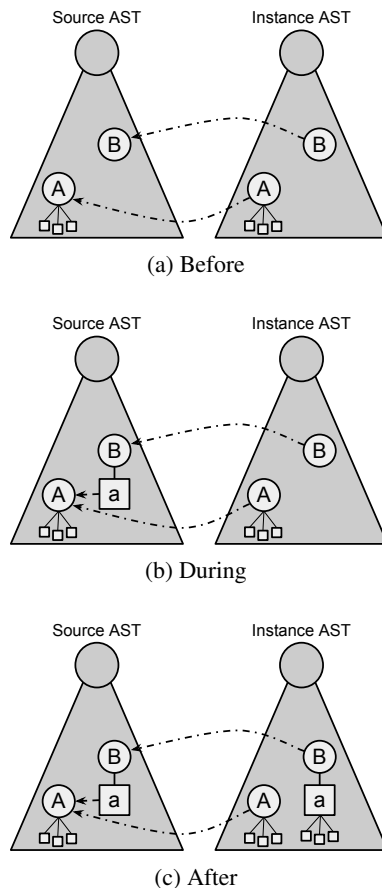


Figure 6: The steps taken when adding a component.

4.4 Preserved Formatting

Consider the graphical editor as currently described in this paper. When the editor changes the model, it does so by modifying the source AST. Eventually these changes need to be displayed in a source code editor or saved to a file. The source AST would then have to be printed back to text format. All information that is significant to the compiler and the graphical editor resides in the AST. It does not, however, traditionally carry any knowledge about how the code was originally formatted. Information that is valuable for the developer, such as indentation and comments would effectively be lost if the AST was simply pretty printed. Somewhere along the way, the information about the original formatting needs to be gathered and stored in the AST so that the original source code can be reprinted.

4.4.1 Scanner and Parser

The *scanner* is the part of the compiler that finds patterns in the code and converts them to a series of tokens given to the *parser*. Those tokens are then con-

verted to a source AST by the parser using the grammar of the language. The issue about preserving formatting, as described in the paragraph above, is solved by letting the scanner add spaces, line breaks and comments to a data structure. Most parentheses are implicit by the Modelica language, but expressions need special care regarding this. The developer might surround expressions by parentheses to explicitly mark precedence. This could, for the sake of readability, be done even when it would already be implicit. The parser collects these parentheses and stores them in respective expression.

When the parsing is finished, a reference to the data structure is added to the nodes in the AST. Later, when the AST is about to be presented in text format, for example, when it should be saved to a file, the information in the data structure is propagated downwards in the AST. At this stage each node gets the formatting related to the node, as the data structure is emptied. After this, the AST is finally printed in text format, where every node has its formatting preserved.

4.4.2 Reading Formatting

As has been mentioned earlier, formatting such as indentation and comments are put in a data structure by the scanner. Some, but not all parentheses should also be collected. The parser has, unlike the scanner, the syntactic information to distinguish which parentheses are significant. That is, which parentheses belong to expressions and which do not. However, the parser has no access to the formatting data structure, so it stores the parentheses directly in the expressions.

When the data structure has been populated by the scanner, it contains a list of *scanned formatting items*. A *formatting item* is an object that contains some basic part of the formatting. It contains the actual string data that should be output when the AST is printed. A formatting item also has information about what type it is, for example *line break* or *comment*. When a formatting item has been scanned it is called a *scanned formatting item*. A scanned formatting item holds the same information as a regular formatting item, but with some information about its origin added. This makes it possible to find its original line and column in the source code. Table 1 shows some typical scanned formatting items.

Before the formatting items in the data structure are used by the AST, some of them are merged. The ones that are adjacent to each other are merged into new, larger formatting items of mixed type. As an example, the two last items in Table 1 would be merged into a

Table 1: The data in some scanned formatting items.

Scanned Formatting Item			
		Formatting Item	
Start	End	Type	Data
(1, 6)	(1, 6)	WHITE_SPACE	" "
(1, 19)	(1, 19)	LINE_BREAK	"\n"
(2, 1)	(2, 4)	WHITE_SPACE	" "
(2, 9)	(2, 9)	WHITE_SPACE	" "
(2, 13)	(2, 13)	WHITE_SPACE	" "
(2, 14)	(2, 29)	COMMENT	"// Text\n"

mixed formatting item. This way each AST node only needs to be associated with one formatting item that can be seen as a prefix, or left-hand side, and one as suffix, right-hand side.

4.4.3 Storing Formatting in the AST

The formatting items are not added to the AST nodes during parsing. Instead they are added during an extra pass when the method for formatted print is called. This approach naturally comes with its advantages and drawbacks.

Approach One of the main reasons for the chosen approach, is to keep the parser as separated from the implementation of this feature as possible. A parser can be complicated enough to begin with. Furthermore, if it turns out that there are any special cases that need to be taken care of, or if parts of the parser need to be rewritten it could become cumbersome and expensive to implement and maintain the code.

Initially, another reason for this approach was that some AST nodes might be *rewritten* and removed when the AST is being accessed. This means that they are replaced with other nodes to make the AST more suitable for semantic analysis [14]. Their formatting would then effectively also disappear. However, this issue is not completely solved by adding the formatting to the AST at a later stage. There are some AST nodes in JModelica.org that are deleted during rewrites that both can come from an explicit keyword or be implied by the Modelica language specification. As an example, members of a Modelica class can be specified to have public or protected visibility⁹, but their default visibility type is public [15]. A public visibility clause is thus an AST node that can come both from a keyword *or* the language specification. This clause is removed during a rewrite, and its child nodes get

⁹Using the keywords `public` and `protected` respectively.

their visibility by assigning them visibility type child nodes. The fact whether the keyword appeared in the source code or not, and if so then where it appeared, still needs to be stored.

The solution presented in this paper comes with a drawback. It is more intense for the CPU to add the formatting to the AST on-demand, rather than in the parser. Firstly, the data structure needs some preparations. Secondly, all AST nodes need to be associated with their corresponding formatting items. This means that the line and column numbers for AST nodes and formatting items need to be compared. It is worth noting, though, that the result from this pass can be cached. This means that consecutive prints of the AST do not need this pass and no more clock cycles are used for this.

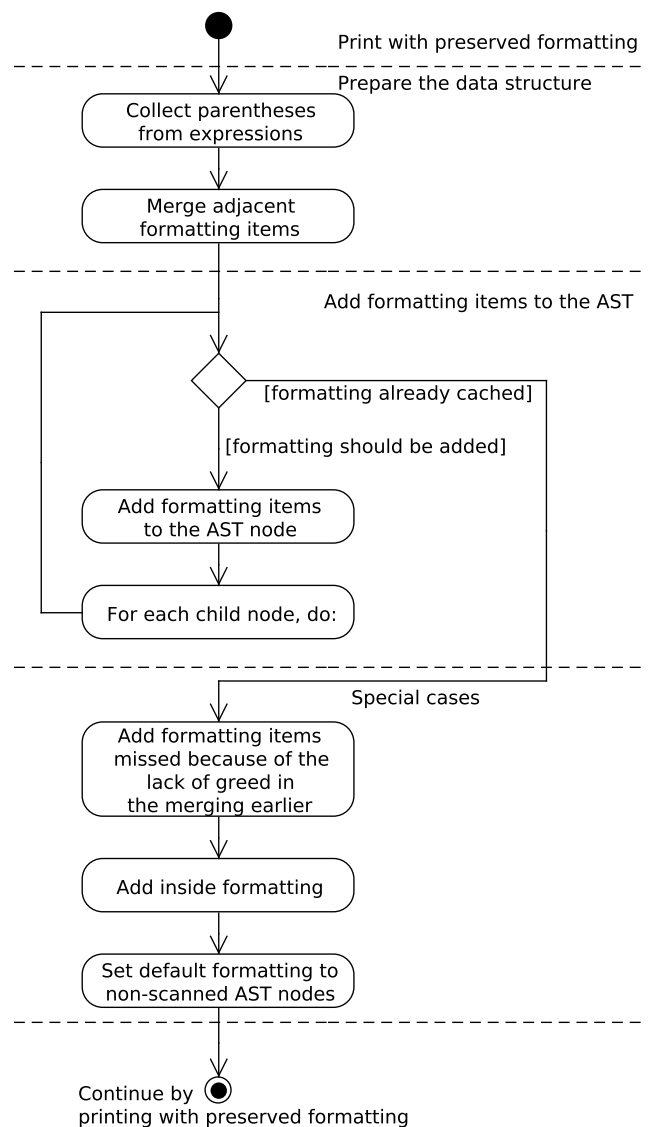


Figure 7: The process for storing formatting in the AST.

Preparations Figure 7 shows the main steps of how the formatting is added to the AST. The first step is to prepare the formatting information from the parsing to be propagated downwards in the tree. The parentheses that were collected to the expressions during parsing are added to the data structure. After that, the formatting items that are adjacent are merged as mentioned in Section 4.4.2. To have a more logical division between the formatting items that should be considered prefix or suffix to an AST node, this algorithm is not *greedy*. This lack of greed means that merged formatting items that span over multiple lines are split into two items on the first line break, instead of always being merged into one. An example of this can be seen in Figure 8. After these preparations, the AST nodes can use the data structure to get their formatting items.

```

model SmallExample
  Real r1; // Regarding r1
  // Comment regarding r2.
  Real r2;
end SmallExample;

```

Legend:

- [-] Prefix formatting
- [] Suffix formatting
- [...] Inside formatting

Figure 8: A screenshot highlighting the different formatting items with boxes. Note that the comments regarding r1 and r2 are not merged into one single mixed item.

Propagation During the propagation, the nodes in the source AST check so that they do not already have a cached result. This happens if the AST already has been reprinted earlier. If they do not have any cached formatting, their formatting is calculated. This is done by going through the data structure filled with formatting items and checking whether they are adjacent to the current AST node, that is, whether their end line and column match the AST node's starting position and vice versa. In this way, each AST node gets two formatting items, one on its left-hand side (prefix) and one on its right-hand side (suffix).

Special Cases If there for some reason are any formatting items left in the data structure when the propagation is done, these still need to find their place in the AST. This can happen, because the merging of adjacent formatting items sometimes generates two formatting items instead of one as described earlier. If

the source code in a file ends with multiple line breaks, only the first one would be added without an extra step.

There are also some AST nodes that contain formatting information inside of them. These nodes usually contain whitespaces at places where they are *atomic*. They are atomic in the sense that they have no child nodes that can use the whitespace as prefix or suffix formatting. These final formatting items left in the data structure are also added.

Finally, a default formatting is set to AST nodes that have been added through another way than during the parsing of the code. Currently, this means AST nodes that have been added by the graphical editor. When this is done, the rest of the implementation is more or less a traditional pretty printer, which of course also prints the formatting information in the AST nodes.

5 Summary and Conclusions

In this paper, an approach for implementing a graphical editor for Modelica built upon the Eclipse framework using GEF, was presented. How GEF can be used to make a clear, yet extensible design for the graphical editor has been discussed. Some of the common pitfalls when integrating systems that describe the same information in different ways, in this case integrating a graphical editor into an existing source code editor, were discussed.

This paper also describes a way to store formatting from an original source in the AST. In the proposed solution, the formatting information is added to the source AST after parsing. Then in a later pass, the information is associated with its corresponding node. Finally, the modified AST can be reprinted with preserved formatting.

The graphical editor in JModelica.org supports basic model editing such as adding, removing and connecting components. Common graphical editing features such as rotation of components and grid snapping are also available. Future development include a parameter dialog for modifying of parameters and improved graphical editing support such as Manhattanized connections.

References

- [1] 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 problem. *Computers and Chemical Engineering*, 34(11):1737–1749, November 2010. Doi:10.1016/j.compchemeng.2009.11.011.
- [2] J. Mattsson, The JModelica IDE: Developing an IDE by Reusing a JastAdd Compiler, Master's thesis, Department of Computer Science, Lund University, Sweden, 2009.
- [3] J. Sten, Graphical Editing in JModelica.org, Master's thesis, Department of Computer Science, Lund University, Sweden, 2012.
- [4] T. Mattsson, AST-driven Editor, Master's thesis, Department of Computer Science, Lund University, Sweden, 2012.
- [5] G. Hedin, E. Magnusson, JastAdd: an aspect-oriented compiler construction system, *Science of Computer Programming* 47 (1) (2003) 37–58. doi:http://dx.doi.org/10.1016/S0167-6423(02)00109-0.
- [6] J. Åkesson. Optimica—an extension of modelica supporting dynamic optimization. In *6th International Modelica Conference 2008*. Modelica Association, March 2008.
- [7] C. Andersson. A new Python-based class for simulation of complex hybrid DAEs and its integration in JModelica.org. Master's thesis, Department of Mathematics, Lund University, Sweden, 2010.
- [8] G. Hedin, "An Introductory Tutorial on JastADD Attribute Grammars," in *Generative and Transformational Techniques in Software Engineering III*, ser. Lecture Notes in Computer Science, vol. 6491. Springer-Verlag Berlin Heidelberg, 2011, pp. 166–200.
- [9] P. Fritzson, A. Pop, K. Norling, M. Blom, Comment- and Indentation Preserving Refactoring and Unparsing for Modelica. In *6th International Modelica Conference 2008*, pp. 657–665. Modelica Association, March 2008.
- [10] M. de Jonge, E. Visser, An Algorithm for Layout Preservation in Refactoring Transformations. *Software Language Engineering*, pp. 40–59, Springer, 2012.
- [11] Eclipse Documentation: ASTRewrite, <http://help.eclipse.org/indigo/topic/org.eclipse.jdt.doc.isv/reference/api/org/eclipse/jdt/core/dom/rewrite/ASTRewrite.html>, 2012.
- [12] S. Asghar, S. Tariq, Design and Implementation of a User Friendly OpenModelica Graphical Connection Editor, Master's thesis, Department of Computer and Information Science, Linköping University, Sweden, 2010.
- [13] S. Asghar, S. Tariq, M. Torabzadeh-Tari, P. Fritzson, A. Pop, M. Sjölund, P. Vasaiely, W. Schamai, An Open Source Modelica Graphic Editor Integrated with Electronic Notebooks and Interactive Simulation. In *8th International Modelica Conference 2011*, pp. 739–747. Modelica Association, March 2011.
- [14] T. Ekman, G. Hedin: Rewritable Reference Attributed Grammars. ECOOP 2004. LNCS, vol. 3086, pp. 147–171. Springer, Heidelberg (2004).
- [15] Modelica Association, Modelica - Language Specification 3.2 rev 1, 2012. <http://modelica.org/documents/ModelicaSpec32Revision1.pdf>.

Model-based Requirement Verification : A Case Study

Feng Liang^{1,2} Wladimir Schamai³ Olena Rogovchenko¹

Sara Sadeghi^{2,4} Mattias Nyberg² Peter Fritzson¹

¹ PELAB - Programming Environment Lab, Dept. Computer Science
Linköping University, SE-581 83 Linköping, Sweden

²Scania, Sweden

³EADS Innovation Works, Engineering
Architecture, 21129 Hamburg, Germany

⁴School of Information and Communication Technology, KTH Royal Institute of Technology

Abstract

This paper presents a complete case study that takes a real Fuel Display System element used in Scania Trucks and applies an unified process for modelling system requirements together with the system itself and verifying these requirements in a structured manner. In order to achieve this process the system is modeled in Modelica, and requirement verification scenarios are specified in ModelicaML and verified with the vVDR (Virtual Verification of Designs against Requirements) approach.

Keywords: system modeling; requirement verification; ModelicaML

1 Introduction

As electronic systems become increasingly complex, so do the requirements that they must fulfill, both in terms of functionality and safety. Thus, maintaining the conformity between the system requirements and the system implementation manually becomes increasingly difficult and unproductive. The goal of this paper is to investigate on the basis of a real case study the integration of modeling based techniques for requirement expression with the actual implementation and the formalization of the requirement verification process.

The case study presented in this paper is a component of a Scania System Model used in real trucks. Scania is one of the leading manufacturers of heavy trucks and buses, operating in over 100 countries with over 35,000 employees and more than 110 years of history.

The Modelica language was chosen to model the system. Modelica is non-proprietary, object-oriented,

equation based language for modeling multi-domain complex physical systems.

2 An Integrated Modeling Approach

2.1 Requirement Specification in the Industrial Context

The presence of Electrical and Electronic(E/E) Systems in vehicles has been increasing rapidly since the early 1970s, coming to cover a wide range of applications. Today's vehicles use around 30 Electronic Control Units (ECUs) for small cars and 80 ECUs for high-end luxury cars and this number keeps growing.

In order to simplify system representation, a concept called SESAMM (Scania Electrical System Architecture Made for Modularization and Maintenance) for SCANIA Truck and Bus electrical systems was developed [6]. However, with this approach the requirements are still kept separate from the system design.

The traditional document-based approach means that all the requirements and design information are written in document form, using natural language and graphics. Although it can be regularized, the document-based approach has fundamental limitations. Traceability and consistency are hard to ensure, since the information is spread out over different documents. Maintenance and reuse are also an issue, and since part of the documents is written in natural language, so is accuracy.

The goal of this work therefore, is to integrate the description of the system requirements into the system modeling process, thus benefitting from all the advantages of model-based engineering.

2.2 ModelicaML

ModelicaML [1] is an UML [2] profile and a language extension for Modelica. The main purpose of ModelicaML is to enable graphical system modeling using the standardized UML notation together with the modeling and simulation power of Modelica. ModelicaML defines different views (e.g., composition, inheritance, behavior) on system models. It is based on a subset of UML and reuses some concepts from SysML. ModelicaML is designed to generate Modelica code from graphical models. Since the ModelicaML profile is an extension of the UML meta-model it can be used as an extension of both UML and SysML. A tool suite for modeling with ModelicaML and generating Modelica code can be downloaded from [3].

2.3 Virtual Verification of Designs against Requirements(vVDR)

vVDR (Virtual Verification of Designs against Requirements) is a method that enables model-based design verification against requirements. The first version of the vVDR method and an example of its application are illustrated in [8] using ModelicaML. ModelicaML supports all Modelica constructs and, in addition, supports an adapted version of the UML state machine and activity diagrams for behavior modeling as well as UML class composition diagrams for structure modeling. This enables engineers to use the simulation power of Modelica combined with a standardized graphical notation for the creation of system models. The main vVDR method steps are:

1. **Formalize Requirements:** This step explains how to formalize requirements for design verification and how to determine which requirements can be verified using this method.
2. **Select or Create Design Model to be verified against Requirements:** This step clarifies what properties a system design model needs to have in order to be suitable for this method.
3. **Select or Create Verification Scenarios:** This step describes what the required properties of a verification scenario are.
4. **Create Verification Models:** This step explains what a verification model consists of and how it can be created.

In order to enable guidance and automation, vVDR introduces the concept of a requirement model, a

design alternative model and a verification scenario. Each of these models is needed in order to create a verification model. In a scenario-based approach, a verification model will comprise one design alternative that is to be verified against a set of requirements by running one verification scenario as illustrated in Figure 1. Moreover, some additional models may be required. For example, a dedicated calculation model might be needed when the required data cannot be provided by the design model if such calculation is not part of the design.

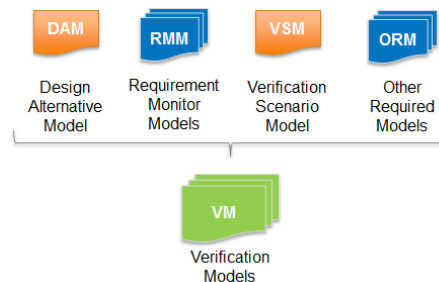


Figure 1: Different models form a Verification Model

Moreover, vVDR anticipates different roles for different tasks, that are most likely to involve different people. Each role requires specific skills and defines the responsibility for different modeling artifacts. For example, the formalization of requirements is performed by a requirements analyst. This person is in charge of requirements elicitation and negotiation. In vVDR this person is also in charge of formalizing the requirements for verification purpose because they are the most familiar with the requirements and, by formalizing them, they will reduce the probability of misinterpretation. The formalization of designs (i.e. the modeling of different design alternatives or versions) is done by the system designer, and the formalization of scenarios as well as the verification itself is done by a tester.

In vVDR a notion of clients, mediators and providers is introduced (see Figure 2). The concept is called Value Bindings [7] and allows capturing of relations that allow determining how different models should be bound when they are combined into verification models. The basic idea for the definition of bindings is the following:

- Each model that requires data from other models should express this need by creating a new mediator or by subscribing to an already existing one.
- Each mediator must have defined providers so that the correct binding code for the clients can

be derived.

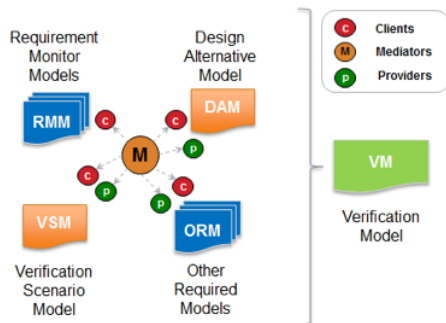


Figure 2: Clients, mediators and providers relations example

The defined relations are used to compose verification models automatically.

3 Methodology

The classification of requirements is important since it affects the requirement selection and verification process. However, there is no consensus in the field of requirement classification. The common sense divides the requirements into functional and non-functional, based on whether they answer the question of “what the system does” or that of “how the system behaves with respect to some observable attributes like performance, usability, maintainability, etc.”, respectively. However, in the practice, it turns out that a more detailed classification of non-functional requirements is needed. Martin Glinz [5] proposed a taxonomy for both functional and non-functional requirements.

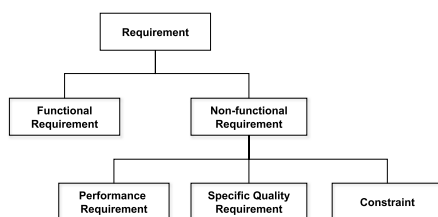


Figure 3: Requirement classification

Figure 3 illustrates the taxonomy proposed by Martin Glinz. In this view, non-functional requirements can be classified as performance requirements, specific quality requirements and constraints. Following is a definition for each category:

Function Requirement is a requirement that describes the system’s reaction to input stimuli.

Performance is a requirement to specify the timing, velocity etc. inside a desired tolerance.

Specific Quality Requirement is a requirement that specifies the quality the system should have like efficiency, security, reliability, usability, maintainability etc.

Constraint is a requirement that constrains the solution space beyond what is necessary for meeting the given function, performance and specific quality requirements.

Not all the requirements are fit to be verified by the simulation model. Some require additional judgment from the stakeholder. For instance, the Specific Quality Requirement which specifies the quality of the system like reliability, maintainability etc., needs to be verified based on the experience of a stakeholder. In contrast to that, the functional and performance requirements which consist of mathematical expressions or boundaries are more suitable to be selected for the dynamic requirement verification process. This is the type of requirement we will concentrate on in this case-study.

3.1 Requirement Formalization

This is the first step of the vVDR method. The main goal of this step is to translate textual requirement statements into formal models that can be processed by computers and determine whether a particular requirement is suited to be verified with this method.

In vVDR, a requirement is formalized by first identifying the quantifiable properties mentioned in the requirement statement and then establishing the relationship between them in order to express when this requirement is evaluated and violated.

3.2 Design Model

In this step, a design model that needs to be verified against requirements is created. Since the design model will be bound with verification scenario and requirement in next steps, it should be able to provide corresponding input to requirement model. This is effectively accomplished by inspecting the mediators that indicate what data is required and of what type the values should be. When building the design model, the modeler associates the providers for each mediator.

In addition to the analysis of the mediators that need providers from the design model at hand, the designer should indicate what the potential stimuli (clients) of this system design model are, that can be set by scenarios (i.e. providers from the scenario models). In

order to do so, the designer subscribes the components that are to be stimulated to existing mediators or creates new mediators respectively. The corresponding providers will be defined in verification scenarios whose creation is explained in the next section. This approach is detailed in [7].

3.3 Verification Scenario

Verification scenarios are models that capture a specific course of actions which stimulate the design model in order to cause a particular reaction. Verification scenarios are created based on requirements with the intention to verify design against requirements. One scenario can be used to verify multiple requirements and one requirement is usually verified using multiple scenarios to increase the confidence in the verification results due to the independence of the scenarios. After creating the verification scenario, it is bound with the designed system and requirements which need to be verified.

3.4 Verification Model Generation

After creating the design model, requirement model and verification scenario, this step is for binding these models in ModelicaML in order to generate executable Modelica code. By using the defined clients, mediators and providers, verification models can now be created automatically by determining valid combinations of scenarios and requirements for a selected system design model.

3.5 Requirement Verification

To express requirement violation, the attribute “status” of type Integer is used, which is created by default for each requirement. The meaning of its value is the following:

- 0 means requirement is not evaluated
- 1 means requirement is evaluated and not violated
- 2 means requirement is evaluated and violated

Now, the verification model generated from ModelicaML in the previous step can be simulated in the Modelica simulation environment. And based on the verification result, the tester will be able to analyse the system design based on the verification result.

4 A Case Study: Fuel Level Display

The case study introduced in this chapter is a Fuel Level Display System (Figure 4), used in Scania Trucks for indicating the fuel level of the truck.



Figure 4: Dash board on Truck

The fuel level system, UF18, has two functionalities:

- fuel level estimation, which is presented as a percentage of the tank that is full. The fuel level should be displayed continuously and work for different vehicle types (truck, bus) and engine types (gas, diesel);
- fuel level warning, which is activated when the fuel level drops below a predefined value, when activated the low level fuel warning should alert the driver by some visible symbol.

These functionalities are represented by two allocation elements, AE201 and AE202 respectively.

4.1 System Architecture

The technical architecture of the Fuel Level Display System is schematized in Figure 5. Three ECUs communicate with each other through CAN-Buses. EMS (Engine Management System) sending the fuel consumption by the engine to COO (Coordinator System) which estimates the fuel level in the tank and evaluates the low fuel level warning. After processing in COO, a signal carries the estimated fuel level in the tank and the low fuel level warning to ICL (Instrument Cluster System). The gauge and bulb in ICL will indicate to the driver how much fuel is left in the tank.

4.2 Requirement Selection and Classification

In Scania, the requirements for UF18, AE201 and AE202 are described in different technical documents

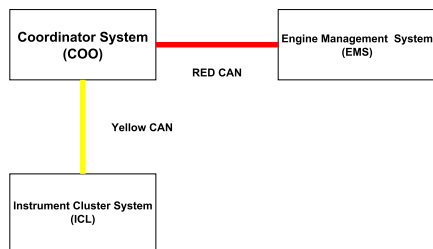


Figure 5: The technical architecture of FLD is composed of three ECUs : Coordinator ECU(COO), Engine Management System(EMS) and Instrument Cluster System(ICL).

respectively. These documents are very extensive, so a subset of elements has been selected for modeling in this case study [4].

<i>UFR18_1</i>	The indicated fuel level shall not deviate more than $\pm 5\%$ from the actual volume in the tank.
<i>UFR18_4</i>	The low fuel level warning shall warn one time when the estimated fuel level reaches below a limit of the measurable volume in the tank. The limit should be 10% for tank sizes below and equal to 900 Liters and 7% for larger tanks.

Based on the requirement classification presented in Section 1, *UFR18_1* is a performance requirement that specifies the tolerance of the indicated fuel level. The other requirement *UFR18_4* is a functional requirement describes how the low fuel level warning behaves with respect to the estimated fuel level.

4.3 Requirement formalization

The next step is to formalize the following requirement statement “*UFR18_1*: The indicated fuel volume shall not deviate more than $\pm 5\%$ from the actual volume in the tank.” The quantifiable properties are the:

- **Indicated fuel volume** (of type Real)
- **Actual volume in tank** (of type Real)
- And the **tolerance** of $\pm\%$ (constant of type Real and the value 0.05)

Note, that there is no precondition that defines when this requirement is valid, i.e., this requirement shall not be violated at any time. A possible precondition could be that this requirement is only valid as long as

the truck is on. In this case the additional quantifiable property identified would be the fact that the truck is on, i.e. “truck is on (of type Boolean)”.

Since this requirement should be checked at all times we only need to express when it is violated or not violated as follows:

```

status = if abs(indicatedFuelLevel
               - actualVolumeInTank) >
           actualVolumeInTank * tolerance
         then 2 else 1
  
```

The code sets the attribute status to 2 (i.e. evaluated and violated) or 1 (evaluated and not violated) depending on whether the absolute value of the difference between the indicated fuel level and actual fuel level in tank is greater than the allowed tolerance or not.

Consider another requirement statement: “*UFR18_4*: The low fuel level warning shall warn one time when the estimated fuel level reaches below a limit of the measurable volume in the tank. The limit should be 10% for tank sizes below and equal to 900 liters and 7% for larger tanks.” The quantifiable properties that are mentioned in this statement are:

- **Estimated fuel level** (of type Real)
- **Warning active** (of type Boolean)
- **Limit** (constant of type Real)
- **Size of the tank** (constant of type Real)

Again, there is no precondition for this requirement so it shall not be violated at any time. To express the violation we could define the status to be:

```

status = if (estimatedFuelLevel
           < sizeOfTank * limit)
         and not warningActive
         then 2 else 1
  
```

All identified properties are inputs that are to be set to the corresponding data from other models, for example the design model or models that capture the design parameters.

4.4 Design Model

In this section, a design model is written in Modelica. Figure 6 illustrates the breakdown of the fuel level display system. It consists of four levels from

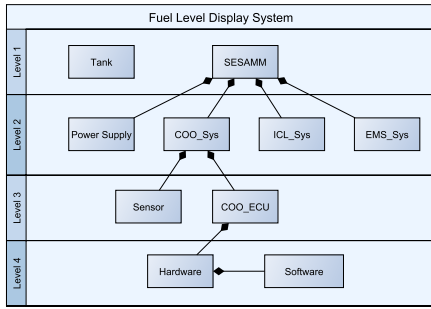


Figure 6: Breakdown of System Design

SESAMM to hardware and software. In the software domain, the application software is implemented in C code generated from the Simulink model through Real-time Workshop(RTW).

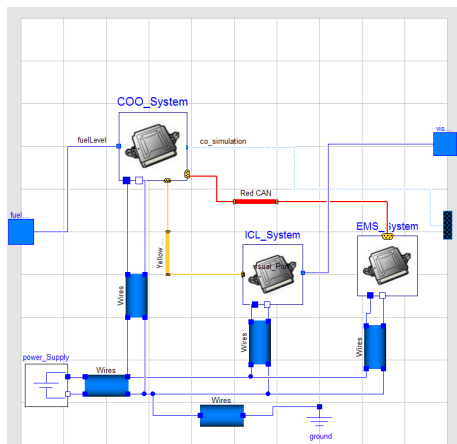


Figure 7: Second level of the system

Figure 7 shows the class diagram of the second level, different ECUs connecting with each other through different CAN-Buses. SESAMM uses different colors of CAN-Buses in order to distinguish between the most safety crucial ECUs and the less safety crucial ECUs. Furthermore, the port located on the top-right of the model carries the indicatedFuelLevel and warningActive calculated by the Simulink model.

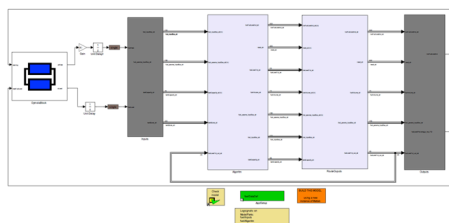


Figure 8: Joint Simulation in Dymola and Simulink

The Design model and the Simulink model are sim-

ulated through a built-in Dymola-Simulink interface as shown in Figure 8. The interface provides the Simulink model with two inputs, fuel rate from the Engine Management System and the fuel level which is measured by a sensor.

4.4.1 Verification Scenario

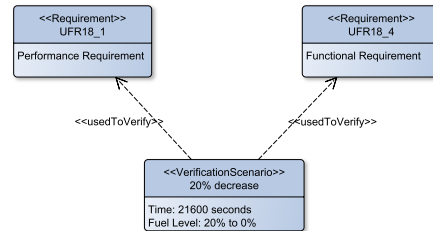


Figure 9: Verification Scenario

After having designed the system, the next step is to create a verification scenario (Figure 9) in order to verify whether the designed system fulfills the requirements. For the fuel level display system, the verification scenario describes how the fuel level in the tank decreases with respect to time. In addition, by inspecting the mediators that represent the need for simulation of the design models, the tester will define corresponding providers that are associated with the mediators.

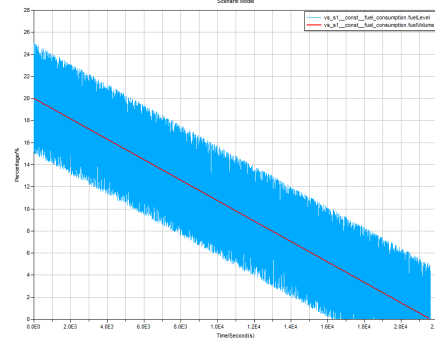


Figure 10: Simulation Result of Scenario Model

In this case study, a verification scenario describes the fuel level in the tank decreasing from 20% to 0% of the capacity of the tank. The verification scenario provides two inputs to the design model, Fuel level and Fuel Volume. Fuel level represents the fuel level measured by the sensor which consists of a noise signal caused by the shaking of tank during driving. The fuel volume represents the ideal fuel volume in the tank. In Figure 10, the blue line represents the fuel level and the red line represents fuel volume. By using this verification scenario, UFR18_1 and UFR18_4 can be verified

at the same time.

4.5 Verification Model Generation

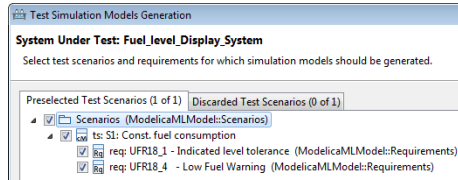


Figure 11: Verification Models Generation

In ModelicaML, the verification model can be generated by binding the design model, to the verification scenario and the requirements. By using the defined clients, mediators and provider verification models can now be created automatically by determining valid combinations of scenarios and requirements for a selected system design alternative model [7] as illustrated in Figure 11. The generated verification models comprise the components that are bound correctly and are ready to be simulated.

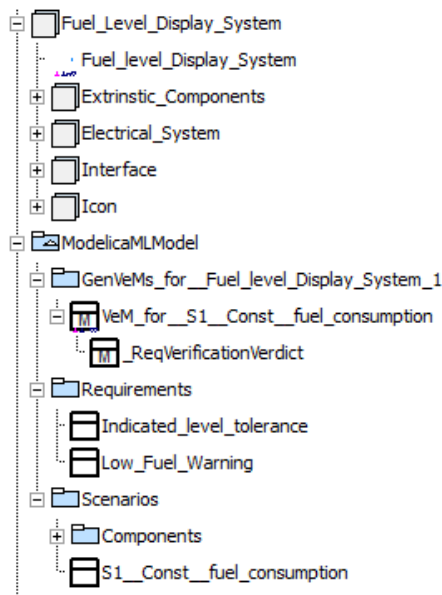


Figure 12: Verification Model in Modelica Simulation Environment

Figure 12 shows the package when the importing verification model to the Modelica simulation environment. The package ModelicaMLModel was created in ModelicaML. It consists of a Verification Model, a Scenario Model and a Requirement Model. The verification model binds other models together and simulates the results.

4.6 Requirement Verification

The verification result of requirement *UFR18_1* is shown in Figure 13 and the verification result of requirement *UFR18_4* is illustrated in Figure 14. As mentioned previously, there is no precondition for these two requirements, so they should be evaluated during the whole verification process.

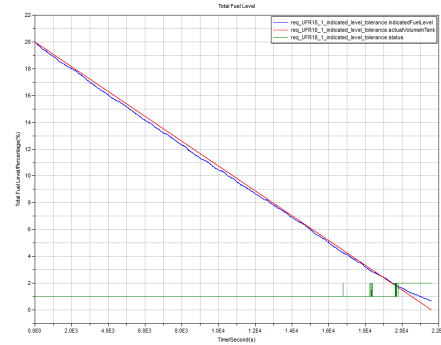


Figure 13: Verification Result of *UFR18_1*

Figure 13 shows the verification result of the Total Fuel Level element. The red line represents the actual volume in the tank and it decreases progressively from 20% to 0%. The blue line shows the indicated fuel level from the Instrument Cluster System. Finally, the green line shows the requirement status. The status starts at 1 which means the requirement is evaluated and not violated until around 20000 seconds. From around 20000 seconds, the status changes to 2 which means that the requirement is evaluated and violated. So the corresponding requirement *UFR18_1* is fulfilled in the first 20000 seconds, then it violated until the end of the simulation.

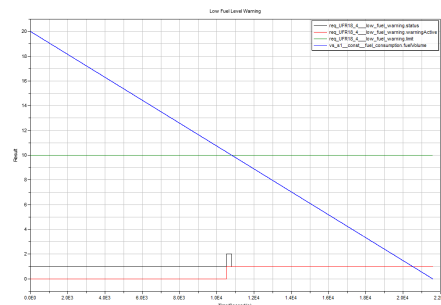


Figure 14: Verification Result of *UFR18_4*

Figure 14 shows the verification results of the Low Fuel Level Warning element. The blue line shows the indicated fuel level from Instrument Cluster System which decreases progressively from 20% to around 0%. The green line shows the threshold at which the low fuel level warning must be enabled. The black

status line illustrates that the requirement UFR18_2 is violated during 10541 second to 10776 second.

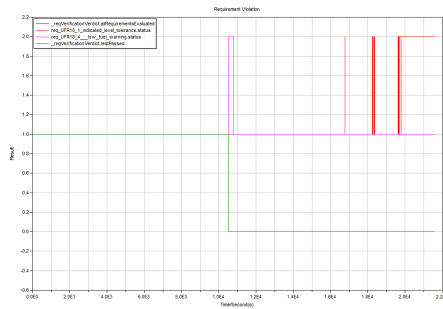


Figure 15: Requirement Violation

Figure 15 is the monitor that shows whether the test is passed or not. As we can see from picture, the test is passed in the first 10541 seconds since both requirements are not violated. After 10541 seconds, requirement UFR18_4 is violated which means that the test fails.

5 Conclusion and Future Work

This case study illustrates the approach to formalizing requirements from document-based format through the vVDR methodology, and generating verification scenarios to test whether the system fulfills these requirements.

The reasons for choosing vVDR approach are its requirements formalization approach, its scalability and the level of possible automation. The way requirements are formalized detects inconsistencies or incompleteness of requirements, it allows expressing requirements monitors using the same formalisms that are used to formalize designs or scenarios, and it allows determining which requirements can be verified using simulations. This is possible based on the knowledge which design models are or will be in place. The generation of verification models, provided by the vVDR approach and its implementation in ModelicaML, automates the process of solving the combinatorial task to select scenarios that are appropriate to stimulate a given design alternative model and all requirements that can be verified by running this scenario. For a small number of requirements, scenarios and design alternatives this approach may be overdone. However, assuming a large number of these artifacts in a real-life project the provided automation is expected to significantly improve the process efficiency.

The goal is to further investigate and generalize the

modelling methodology in the industrial context, by applying to larger test cases and formalizing the process. This work is part of a larger project on an integrated toolchain from documentation formalization through to requirement verification and fault tolerance analysis.

6 Acknowledgement

This work has been supported by Scania, by the EL-LIIT project, the Swedish Strategic Research Foundation in the EDop and HIPO projects, and Vinnova in the RTSIM and ITEA2 OPENPROD projects. The Open Source Modelica Consortium supports the OpenModelica work.

References

- [1] Modelica Association: Modelica: A Unified Object-Oriented Language for Physical Systems Modeling: Language Specification Version 3.2. <http://www.modelica.org>.
- [2] Object Management Group: OMG Unified Modeling Language TM (OMG UML). <http://www.uml.org/>.
- [3] OpenModelica Project: ModelicaML - A UML Profile for Modelica. <http://www.openmodelica.org/modelicaml>.
- [4] C. Erlandsson. Revising the user function requirements document of fuel level display for compliance with iso 26262 and literature. Master's thesis, 2012.
- [5] Martin Glinz. On non-functional requirements. In *Requirements Engineering Conference*, pages 21–26, 2007.
- [6] L. Hans. The SESAMM concept. PD1422538, 2003.
- [7] Wladimir Schamai, Peter Fritzson, Christiaan J. J. Paredis, and Philipp Helle. Modelicaml value bindings for automated model composition. In *Proc. of Symposium on Theory of Modeling and Simulation (TMS/DEVS 2012)*, 2012.
- [8] Wladimir Schamai, Philipp Helle, Peter Fritzson, and Christiaan J. J. Paredis. Virtual verification of system designs against system requirements. In *MoDELS Workshops*, pages 75–89, 2010.

A Data-Parallel Algorithmic Modelica Extension for Efficient Execution on Multi-Core Platforms

Mahder Gebremedhin, Afshin Hemmati Moghadam, Peter Fritzson, Kristian Stavåker
Department of Computer and Information Science
Linköping University, SE-581 83 Linköping, Sweden
{mahder.gebremedin, peter.fritzson, Kristian.stavaker}@liu.se, afshe586@student.liu.se

Abstract

New multi-core CPU and GPU architectures promise high computational power at a low cost if suitable computational algorithms can be developed. However, parallel programming for such architectures is usually non-portable, low-level and error-prone. To make the computational power of new multi-core architectures more easily available to Modelica modelers, we have developed the ParModelica algorithmic language extension to the high-level Modelica modeling language, together with a prototype implementation in the OpenModelica framework. This enables the Modelica modeler to express parallel algorithms directly at the Modelica language level. The generated code is portable between several multi-core architectures since it is based on the OpenCL programming model. The implementation has been evaluated on a benchmark suite containing models with matrix multiplication, Eigen value computation, and stationary heat conduction. Good speedups were obtained for large problem sizes on both multi-core CPUs and GPUs. To our knowledge, this is the first high-performing portable explicit parallel programming extension to Modelica.

Keywords: Parallel, Simulation, Benchmarking, Modelica, Compiler, GPU, OpenCL, Multi-Core

1 Introduction

Models of large industrial systems are becoming increasingly complex, causing long computation time for simulation. This makes it attractive to investigate methods to use modern multi-core architectures to speedup computations.

Efficient parallel execution of Modelica models has been a research goal of our group for a long time [4], [5], [6], [7], involving improvements both in the compilation process and in the run-time system for parallel execution. Our previous work on compilation of data-parallel models, [7] and [8], has primarily addressed

compilation of purely equation-based Modelica models for simulation on NVIDIA Graphic Processing Units (GPUs). Several parallel architectures have been targeted, such as standard Intel multi-core CPUs, IBM Cell B.E, and NVIDIA GPUs. All the implementation work has been done in the OpenModelica compiler framework [2], which is an open-source implementation of a Modelica compiler, simulator, and development environment. Related research on parallel numeric solvers can for example be found in [9].

The work presented in this paper presents an algorithmic Modelica language extension called ParModelica for efficient portable explicit parallel Modelica programming. Portability is achieved based on the OpenCL [14] standard which is available on several multi-core architectures. ParModelica is evaluated using a benchmark test suite called Modelica PARallel benchmark suite (MPAR) which makes use of these language extensions and includes models which represent heavy computations.

This paper is organized as follows. Section 2 gives a general introduction to Modelica simulation on parallel architectures. Section 3 gives an overview of GPUs, CUDA and OpenCL, whereas the new parallel Modelica language extensions are presented in Section 4. Section 5 briefly describes measurements using the parallel benchmark test suite. Finally, Section 6 gives programming guidelines to use ParModelica, and Section 7 presents conclusions and future work.

2 Parallel Simulation of Modelica Models on Multi-Core Computers

The process of compiling and simulating Modelica models to sequential code is described e.g. in [3] and [12]. The handling of equations is rather complex and involves symbolic index reduction, topological sorting according to the causal dependencies between the equations, conversion into assignment statement form, etc. Simulation corresponds to "solving" the compiled

equation system with respect to time using a numerical integration method.

Compiling Modelica models for efficient parallel simulation on multi-core architectures requires additional methods compared to the typical approaches described in [3] and [12]. The parallel methods can be roughly divided into the following three groups:

- *Automatic parallelization of Modelica models.* Several approaches have been investigated: centralized solver approach, distributed solver approach and compilation of unexpanded array equations. With the first approach the solver is run on one core and in each time-step the computation of the equation system is done in parallel over several cores [4]. In the second approach the solver and the equation system are distributed across several cores [5]. With the third approach Modelica models with array equations are compiled unexpanded and simulated on multi-core architectures.
- *Coarse-grained explicit parallelization using components.* Components of the model are simulated in parallel partly de-coupled using time delays between the different components, see [11] for a summary. A different solver, with different time step, etc., can be used for each component. A related approach has been used in the xMOD tool [26].
- *Explicit parallel programming language constructs.* This approach is explored in the NestStepModelica prototype [10] and in this paper with the ParModelica language extension. Parallel extensions have been developed for other languages, e.g. parfor loop and gpu arrays in Matlab, Visual C++ parallel_for, Mathematica parallelDo, etc.

3 GPU Architectures, CUDA, and OpenCL

Graphics Processing Units (GPUs) have recently become increasingly programmable and applicable to general purpose numeric computing. The theoretical processing power of GPUs has in recent years far surpassed that of CPUs due to the highly parallel computing approach of GPUs.

However, to get good performance, GPU architectures should be used for simulation of models of a regular structure with large numbers of similar data objects. The computations related to each data object can then be executed in parallel, one or more data objects on each core, so-called data-parallel computing. It is also very important to use the GPU memory hierarchy effectively in order to get good performance.

In Section 3.1 the NVIDIA GPU with its CUDA programming model is presented as an influential example of GPU architecture, followed by the portable OpenCL parallel programming model in Section 3.2.

3.1 NVIDIA GPU CUDA – Compute Unified Device Architecture

An important concept in NVIDIA CUDA (Computer Unified Device Architecture) for GPU programming is the distinction between host and device. The host is what executes normal programs, and the device works as a coprocessor to the host which runs CUDA threads by instruction from the host. This typically means that a CPU is the host and a GPU is the device, but it is also possible to debug CUDA programs by using the CPU as both host and device. The host and the device are assumed to have their own separate address spaces, the host memory and the device memory. The host can use the CUDA runtime API to control the device, for example to allocate memory on the device and to transfer memory to and from the device.

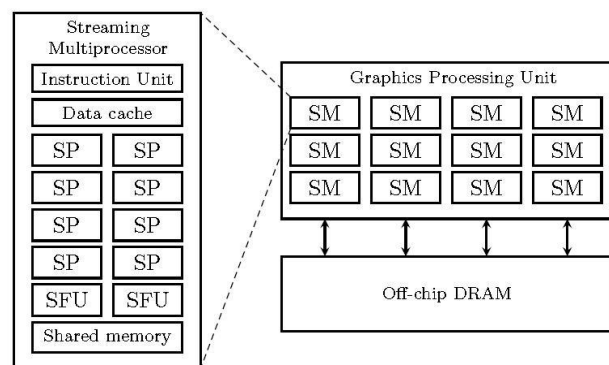


Figure 1. Simplified schematic of NVIDIA GPU architecture, consisting of a set of Streaming Multiprocessors (SM), each containing a number of Scalar Processors (SP) with fast private memory and on-chip local shared memory. The GPU also has off-chip DRAM.

The building block of the NVIDIA CUDA hardware architecture is the *Streaming Multiprocessor* (SM). In the NVIDIA Fermi-Tesla M2050 GPU, each SM contains 32 *Scalar Processors* (SPs). The entire GPU has 14 such SMs totaling to 448 SPs, as well as some off-chip DRAM memory, see Figure 1. This gives a scalable architecture where the performance of the GPU can be varied by having more or fewer SMs.

To be able to take advantage of this architecture a program meant to run on the GPU, known as a *kernel*, needs to be massively multi-threaded. A kernel is just a C-function meant to execute on the GPU. When a kernel is executed on the GPU it is divided into *thread blocks*, where each thread block contains an equal number of threads. These thread blocks are automatically distributed among the SMs, so a programmer

need not consider the number of SMs a certain GPU has. All threads execute one common instruction at a time. If any threads take divergent execution paths, then each of these paths will be executed separately, and the threads will then converge again when all paths have been executed. This means that some SPs will be idle if the thread executions diverge. It is thus important that all threads agree on an execution path for optimal performance.

This architecture is similar to the *Single Instruction, Multiple Data* (SIMD) architecture that vector processors use, and that most modern general-purpose CPUs have limited capabilities for too. NVIDIA call this architecture *Single Instruction, Multiple Thread* (SIMT) instead, the difference being that each thread can execute independently, although at the cost of reduced performance. It is also possible to regard each SM as a separate processor, which enables Multiple Instructions, Multiple Data (MIMD) parallelism. Using only MIMD parallelism will not make it possible to take full advantage of a GPU's power, since each SM is a SIMD processor. To summarize:

- Streaming Multiprocessors (SM) can work with different code, performing different operations with entirely different data (MIMD execution, Multiple Instruction Multiple Data).
- All Scalar processors (SP) in one streaming multiprocessor execute the same instruction at the same time but work on different data (SIMT/SIMD execution, Single Instruction Multiple Data).

3.1.1 NVIDIA GPU Memory Hierarchy

As can be seen in Figure 1 there are several different types of memory in the CUDA hardware architecture. At the lowest level each SP has a set of registers, the number depending on the GPU's capabilities. These registers are shared between all threads allocated to a SM, so the number of thread blocks that a SM can have active at the same time is limited by the register usage of each thread. Accessing a register typically requires no extra clock cycles per instruction, except for some special cases where delays may occur.

Besides the registers there is also the shared (local) memory, which is shared by all SPs in a SM. The shared memory is implemented as fast on-chip memory, and accessing the shared memory is generally as fast as accessing a register. Since the shared memory is accessible to all threads in a block it allows the threads to cooperate efficiently by giving them fast access to the same data.

Most of the GPU memory is off-chip Dynamic Random Access Memory (DRAM). The amount of off-

chip memory on modern graphics cards range from several hundred megabytes to few gigabytes. The DRAM memory is much slower than the on-chip memories, and is also the only memory that is accessible to the host CPU, e.g. through DMA transfers. To summarize:

- Each scalar processor (SP) has a set of fast registers. (private memory)
- Each streaming multiprocessor (SM) has a small local shared memory (48KB on Tesla M2050) with relatively fast access.
- Each GPU device has a slower off-chip DRAM (2GB on Tesla M2050) which is accessible from all streaming multiprocessors and externally e.g. from the CPU with DMA transfers.

3.2 OpenCL – the Open Computing Language

OpenCL [14] is the first open, free parallel computing standard for cross-platform parallel programming of modern processors including GPUs. The OpenCL programming language is based on C99 with some extensions for parallel execution management. By using OpenCL it is possible to write parallel algorithms that can be easily ported between multiple devices with minimal or no changes to the source code.

The OpenCL framework consists of the OpenCL programming language, API, libraries, and a runtime system to support software development. The framework can be divided into a hierarchy of models: Platform Model, Memory model, Execution model, and Programming model.

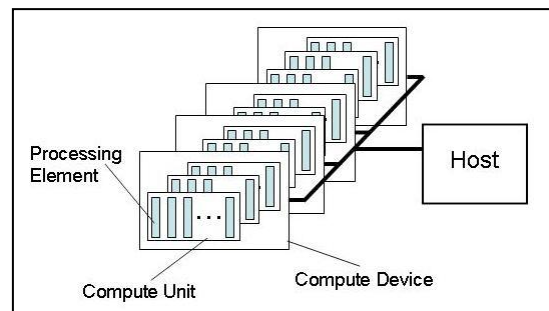


Figure 2. OpenCL platform architecture.

The OpenCL platform architecture in Figure 2 is similar to the NVIDIA CUDA architecture in Figure 1:

- Compute device – Graphics Processing Unit (GPU)
- Compute unit – Streaming Multiprocessor (SM)
- Processing element – Scalar Processor (SP)
- Work-item – thread
- Work-group – thread block

The memory hierarchy (Figure 3) is also very similar:

- Global memory – GPU off-chip DRAM memory

- Constant memory – read-only cache of off-chip memory
- Local memory – on-chip shared memory that can be accessed by threads in the same SM
- Private memory – on-chip registers in the same

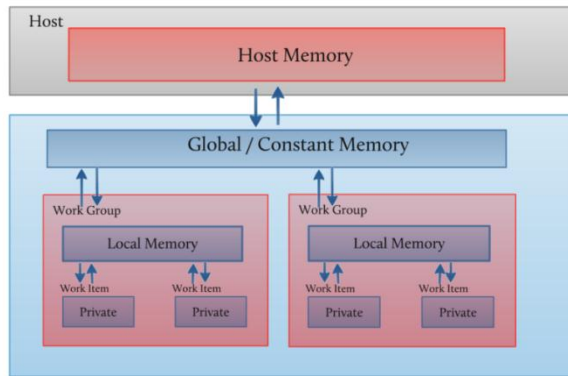


Figure 3. Memory hierarchy in the OpenCL memory model, closely related to typical GPU architectures such as NVIDIA.

The memory regions can be accessed in the following way:

Memory Regions	Access to Memory
Constant Memory	All work-items in all work-groups
Local Memory	All work-items in a work-group
Private Memory	Private to a work-item
Global Memory	All work-items in all work-groups

3.2.1 OpenCL Execution Model

The execution of an OpenCL program consists of two parts, the *host program* which executes on the host and the parallel *OpenCL program*, i.e., a collection of *kernels* (also called kernel functions), which execute on the OpenCL device. The host program manages the execution of the OpenCL program.

Kernels are executed simultaneously by all threads specified for the kernel execution. The number and mapping of threads to Computing Units of the OpenCL device is handled by the host program.

Each *thread* executing an instance of a kernel is called a *work-item*. Each thread or work item has *unique id* to help identify it. Work items can have additional id fields depending on the arrangement specified by the host program.

Work-items can be arranged into *work-groups*. Each work-group has a unique ID. Work-items are assigned a unique local ID within a work-group so that a single work-item can be uniquely identified by its global ID or by a combination of its local ID and work-group ID.

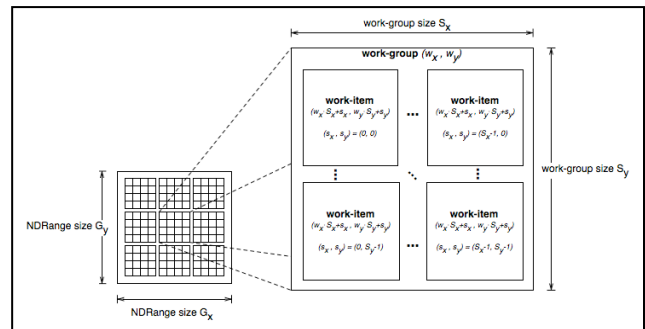


Figure 4. OpenCL execution model, work-groups depicted as groups of squares corresponding to work-items. Each work-group can be referred to by a unique ID, and each work-item by a unique local ID.

The work-items in a given work-group execute concurrently on the processing elements of a single compute unit as depicted in Figure 4.

Several programming models can be mapped onto this execution model. OpenCL explicitly supports two of these models: primarily the data parallel programming model, but also the task parallel programming model

4 ParModelica: Extending Modelica for Explicit Algorithmic Parallel Programming

As mentioned in the introduction, the focus of the current work is an extension (ParModelica) of the algorithmic subset of Modelica for efficient explicit parallel programming on highly data-parallel SPMD (Single Program Multiple Data) architectures. The current ParModelica implementation generates OpenCL [14] code for parallel algorithms. OpenCL was selected instead of CUDA [15] because of its portability between several multi-core platforms. Generating OpenCL code ensures that simulations can be run with parallel support on OpenCL enabled Graphics and Central Processor Units (GPUs and CPUs). This includes many multi-core CPUs from [19] and Advanced Micro Devices (AMD) [18] as well as a range of GPUs from NVIDIA [17] and AMD [18].

As mentioned earlier most previous work regarding parallel execution support in the OpenModelica compiler has been focused on automatic parallelization where the burden of finding and analyzing parallelism has been put on the compiler. In this work, however, we have decided to leave this responsibility to the end user programmer. The compiler provides additional high level language constructs needed for explicitly stating parallelism in the algorithmic part of the modeling language. These, among others, include parallel variables, parallel functions, kernel functions and paral-

lel for loops indicated by the parfor keyword. There are also some target language specific constructs and functions (in this case related to OpenCL).

4.1 Parallel Variables

OpenCL code can be executed on a host CPU as well as on GPUs whereas CUDA code executes only on GPUs. Since the OpenCL and CUDA enabled GPUs use their own local (different from CPU) memory for execution, all necessary data should be copied to the specific device's memory. Parallel variables are allocated on the specific device memory instead of the host CPU. An example is shown below:

```
function parvar
protected
  Integer m = 1000;           // Host Scalar
  Integer A[m,m];           // Host Matrix
  Integer B[m,m];           // Host Matrix
// global and local device memories
  parglobal Integer pm;     // Global Scalar
  parglobal Integer pA[m,m]; // Glob Matrix
  parglobal Integer pB[m,m]; // Glob Matrix
  parlocal Integer pn;     // Local Scalar
  parlocal Integer pS[m];  // Local Array
end parvar;
```

The first two matrices A and B are allocated in normal host memory. The next two matrices pA and pB are allocated on the global memory space of the OpenCL device to be used for execution. These global variables can be initialized from normal or host variables. The last array pS is allocated in the local memory space of each processor on the OpenCL device. These variables are shared between threads in a single work-group and cannot be initialized from host variables.

Copying of data between the host memory and the device memory used for parallel execution is as simple as *assigning the variables* to each other. The compiler and the runtime system handle the details of the operation. The assignments below are all valid in the function given above

- Normal assignment - $A := B$
- Copy from host memory to parallel execution device memory - $pA := A$
- Copy from parallel execution device memory to host memory - $B := pB$
- Copy from device memory to other device memory - $pA := pB$

Modelica parallel arrays are passed to functions only by reference. This is done to reduce the rather expensive copy operations.

4.2 Parallel Functions

ParModelica *parallel functions* correspond to OpenCL functions *defined in kernel files* or to CUDA device

functions. These are functions available for distributed (parallel) independent execution in each thread executing on the parallel device. For example, if a parallel array has been distributed with one element in each thread, a parallel function may operate locally in parallel on each element. However, unlike kernel functions, *parallel functions cannot be called from serial code* in normal Modelica functions on the host computer just as parallel OpenCL functions are not allowed to be called from serial C code on the host. Parallel functions have the following constraints, primarily since they are assumed to be called within a parallel context in work-items:

- Parallel function bodies may *not* contain parfor-loops. The reason is that the kernel containing the parallel functions is already distributed on each thread.
- Explicitly declared parallel variables are not allowed since execution is already taking place on the parallel device.
- All memory allocation will be on the parallel device's memory.
- Nested parallelism as in NestStepModelica [10] is not supported by this implementation.
- Called functions must be parallel functions or supported built-in functions since execution is on the parallel device.
- Parallel functions can only be called from the body of a parfor-loop, from parallel functions, or from kernel functions.

Parallel functions in ParModelica are defined in the same way as normal Modelica functions, except that they are preceded by the `parallel` keyword as in the multiply function below:

```
parallel function multiply
  input parglobal Integer a;
  input parlocal Integer b;
  output parprivate Integer c; // same as
output Integer c;
algorithm
  c := a * b;
end multiply;
```

4.3 Kernel Functions

ParModelica *kernel functions* correspond to OpenCL kernel functions [14] or CUDA global functions [16]. They are simply functions compiled to execute on an OpenCL parallel device, typically a GPU. ParModelica kernel functions are allowed to have several return- or output variables unlike their OpenCL or CUDA counterparts. They can also allocate memory in the global address space. Kernel functions can be called from serial host code, and are executed by each thread in the

launch of the kernel. Kernel functions share the first three constraints stated above for parallel functions.

However, unlike parallel functions, kernel functions *cannot* be called from the body of a parfor-loop or from other kernel functions.

Kernel functions in ParModelica are defined in the same way as normal Modelica functions, except that they are preceded by the `kernel` keyword. An example usage of kernel functions is shown by the kernel function `arrayElemWiseMult`. The thread id function `oclGetGlobalId()` (see Section 4.5) returns the integer id of a work-item in the first dimension of a work group.

```
kernel function arrayElemWiseMultiply
input Integer m;
input Integer A[m];
input Integer B[m];
output Integer C[m];
protected
Integer id;
algorithm
id := oclGetGlobalId(1);
// calling the parallel function
multiply is OK from kernel functions
C[id] := multiply(A[id],B[id]); //
multiply can be replaced by A[id]*B[id]
end arrayElemWiseMultiply;
```

4.4 Parallel For Loop: parfor

The iterations of a ParModelica parfor-loop are executed without any specific order in *parallel* and *independently* by multiple threads. The iterations of a parfor-loop are equally distributed among available processing units. If the range of the iteration is smaller than or equal to the number of threads the parallel device supports, each iteration will be done by a separate thread. If the number of iterations is larger than the number of threads available, some threads might perform more than one iteration. In future enhancements parfor will be given the extra feature of specifying the desired number of threads explicitly instead of automatically launching threads as described above. An example of using the parfor-loop is shown below:

```
// Matrix multiplication using parfor loop
parfor i in 1:m loop
for j in 1:pm loop
ptemp := 0;
for h in 1:pm loop // calling the
// parallel function multiply is OK
// from parfor-loops
ptemp := multiply(pA[i,h], pB[h,j])
+ ptemp;
end for;
pC[i,j] := ptemp;
end for;
end parfor;
```

ParModelica parallel for loops, compared to normal Modelica for loops, have some additional constraints:

- All variable references in the loop body must be to parallel variables.
- Iterations should not be dependent on other iterations i.e. no loop-carried dependencies.
- All function calls in the body should be to parallel functions or supported built-in functions only.

4.5 Executing User-written OpenCL Code from ParModelica.

There are also some additional ParModelica features available for directly compiling and executing user-written OpenCL code:

- `oclbuild(String)` takes a name of an OpenCL source file and builds it. It returns an OpenCL program object which can be used later.
- `oclkernel(oclprogram, String)` takes a previously built OpenCL program and create the kernel specified by the second argument. It returns an OpenCL kernel object which can be used later.
- `oclsetargs(oclkernel,...)` takes a previously created kernel object variable and a variable number of arguments and sets each argument to its corresponding one in the kernel definition.
- `oclexecute(oclkernel)` executes the specified kernel.

All of the above operations are synchronous in the OpenCL jargon. They will return only when the specified operation is completed. Further functionality is planned to be added to these functions to provide better control over execution.

4.6 Synchronization and Thread Management

All OpenCL work-item functions [20] are available in ParModelica. They perform the same operations and have the “same” types and number of arguments. However, there are two main differences:

- Thread/work-item index ids start from 1 in ParModelica, whereas the OpenCL C implementation counts from 0.
- Array dimensions start from 1 in Modelica and from 0 in OpenCL and C.

For example `oclGetGlobalId(1)` call in the above `arrayElemWiseMultiply` will return the integer ID of a work-item or thread in the first dimension of a work group. The first thread gets an ID of 1. The OpenCL C call for the same operation would be `ocl_get_global_id(0)` with the first thread obtaining an ID of 0.

In addition to the above features, special built-in functions for building user written OpenCL code directly from source code, creating a kernel, setting arguments to kernel and execution of kernels are also available. In addition parallel versions of some built-in algorithm functions are also available.

5 Benchmarking and Evaluation

To be able to evaluate the relative performance and behavior of the new language extensions described in Section 4, performing systematic benchmarking on a set of appropriate Modelica models is required. For this purpose we have constructed a benchmark test suite containing some models that represent heavy and high-performance computation, relevant for simulation on parallel architectures.

5.1 The MPAR Benchmark Suite

The MPAR benchmark test suite contains seven different algorithms from well-known benchmark applications such as the LINear equations software PACKage (LINPACK) [21], and Heat Conduction [23]. These benchmarks have been collected and implemented as algorithmic time-independent Modelica models.

The algorithms implemented in this suite involve rather large computations and impose well defined workloads on the OpenModelica compiler and the run-time system. Moreover, they include different kinds of for-loops and function calls which provide parallelism for domain and task decomposition. For space reasons we have provided results for only three models here.

Time measurements have been performed of both sequential and parallel implementations of three models: Matrix Multiplication, Eigen value computation, and Stationary Heat Conduction, on both CPU and GPU architectures. For executing sequential codes generated by the standard sequential OpenModelica compiler we have used the Intel Xeon E5520 CPU [24] which has 16 cores, each with 2.27 GHz clock frequency. For executing generated code by our new OpenCL based parallel code generator, we have used the same CPU as well as the NVIDIA Fermi-Tesla M2050 GPU [25].

5.2 Measurements

In this section we present the result of measurements for simulating three models from the implemented benchmark suite. On each hardware configuration all simulations are performed five times with start time 0.0, stop time of 0.2 seconds and 0.2 seconds time step, measuring the average simulation time using the `clock_gettime()` function from the C standard li-

brary. This function is called once when the simulation loop starts and once when the simulation loop finishes. The difference between the returned values gives the simulation time.

All benchmarks have been simulated on both the Intel Xeon E5520 CPU (16 cores) and the NVIDIA Fermi-Tesla M2050 GPU (448 cores).

5.3 Simulation Results

The Matrix Multiplication model (Appendix A) produces an $M \times K$ matrix C from multiplying an $M \times N$ matrix A by an $N \times K$ matrix B . This model presents a very large level of data-parallelism for which a considerable speedup has been achieved as a result of parallel simulation of this model on parallel platforms. The simulation results are illustrated in Figure 5 and Figure 6. The obtained speedup of matrix multiplication using kernel functions is as follows compared to the sequential algorithm on Intel Xeon E5520 CPU:

- Intel 16-core CPU – speedup 26
- NVIDIA 448-core GPU – speedup 115

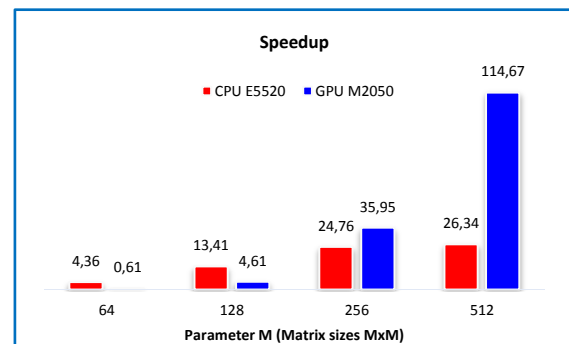


Figure 5. Speedup for matrix multiplication, Intel 16-core CPU and Nvidia 448 core GPU.

The measured matrix multiplication model simulation times can be found in Figure 6.

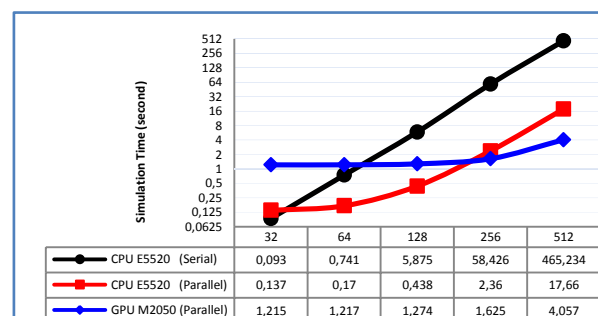


Figure 6. Simulation time for matrix multiplication, Intel 1-core, 16-core CPU, Nvidia 448 core GPU.

The second benchmark model performs Eigen-value computation, with the following speedups:

- Intel 16-core CPU – speedup 3

- NVIDIA 448-core GPU – speedup 48

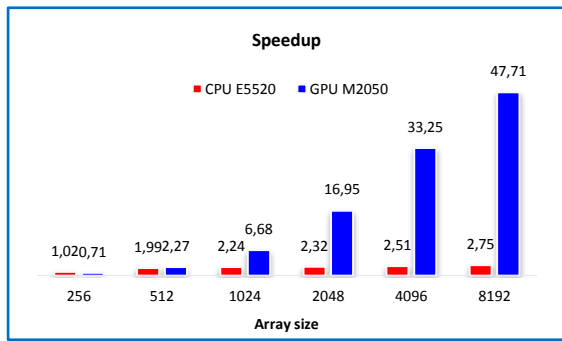


Figure 7. Speedup for Eigen value computation as a function of model array size, for Intel 16-core CPU and NVIDIA 448 core GPU, compared to the sequential algorithm on Intel Xeon E5520 CPU.

The measured simulation times for the Eigen-value model are shown in Figure 8.

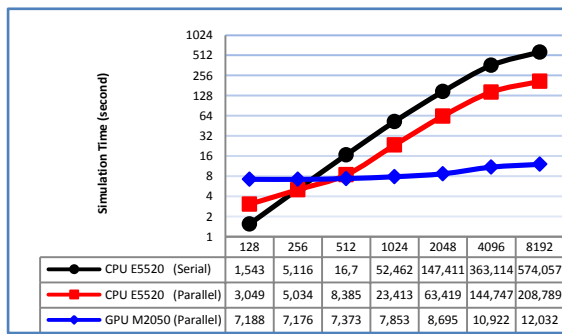


Figure 8. Simulation time for Eigen-value computation as a function of model array size, for Intel 1-core CPU, 16-core CPU, and NVIDIA 448 core GPU.

The third benchmark model computes stationary heat conduction, with the following speedups:

- Intel 16-core CPU – speedup 7
- NVIDIA 448-core GPU – speedup 22

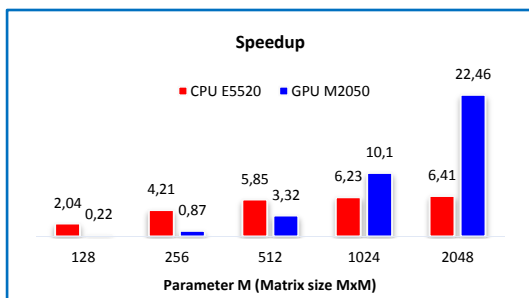


Figure 9. Speedup for the heat conduction model as a function of model size parameter M, Intel 16-core CPU and Nvidia 448 core GPU, compared to sequential algorithm on Intel Xeon E5520 CPU.

The measured simulation times for the stationary heat conduction model are shown in Figure 10.

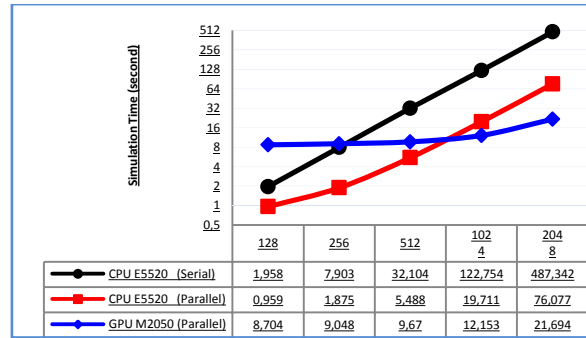


Figure 10. Simulation time (seconds) for heat conduction model as a function of model size parameter M, for 1-core CPU, 16-core CPU, and 448 core GPU.

According to the results of our measurements illustrated in Figure 5, Figure 7, and Figure 9, absolute speedups of 114, 48, and 22 respectively were achieved when running generated ParModelica OpenCL code on the Fermi-Tesla M2050 GPU compared to serial code on the Intel Xeon E5520 CPU with the largest data sizes.

It should be noted that when the problem size is not very large the sequential execution has better performance than the parallel execution. This is not surprising since for executing even a simple code on OpenCL devices it is required to create an OpenCL context within those devices, allocate OpenCL memory objects, transfer input data from host to those memory objects, perform computations, and finally transfer back the result to the host. Consequently, performing all these operations normally takes more time compared to the sequential execution when the problem size is small.

It can also be seen that, as the sizes of the models increase, the simulations get better relative performance on the GPU compared to multi-core CPU. Thus, to fully utilize the power of parallelism using GPUs it is required to have large regular data structures which can be operated on simultaneously by being decomposed to all blocks and threads available on GPU. Otherwise, executing parallel codes on a multi-core CPU would be a better choice than a GPU to achieve more efficiency and speedup.

6 Guidelines for Using the New Parallel Language Constructs

The most important task in all approaches regarding parallel code generation is to provide an appropriate way for analyzing and finding parallelism in sequential codes. In automatic parallelization approaches, the whole burden of this task is on the compiler and tool developer. However, in explicit parallelization approaches as in this paper, it is the responsibility of the modeler to analyze the source code and define which

parts of the code are more appropriate to be explicitly parallelized. This requires a good understanding of the concepts of parallelism to avoid inefficient and incorrect generated code. In addition, it is necessary to know the constraints and limitations involved with using explicit parallel language constructs to avoid compile time errors. Therefore we give some advice on how to use the ParModelica language extensions to parallelize Modelica models efficiently:

- Try to declare parallel variables as well as copy assignments among normal and parallel variables as less as possible since the costs of data transfers from host to devices and vice versa are very expensive.
- In order to minimize the number of parallel variables as well as data transfers between host and devices, it is better not to convert for-loops with few iterations over simple operations to parallel for-loops (parfor-loops).
- It is not always useful to have parallel variables and parfor-loops in the body of a normal for-loop which has many iterations. Especially in cases where there are many copy assignments among normal and parallel variables.
- Although it is possible to declare parallel variables and also parfor-loops in a function, there are no advantages when there are many calls to the function (especially in the body of a big for-loop). This will increase the number of memory allocations for parallel variables as well as the number of expensive copies required to transfer data between host and devices.
- Do not directly convert a for-loop to a parfor-loop when the result of each iteration depends on other iterations. In this case, although the compiler will correctly generate parallel code for the loop, the result of the computation may be incorrect.
- Use a parfor-loop in situations where the loop has many independent iterations and each iteration takes a long time to be completed.
- Try to parallelize models using kernel functions as much as possible rather than using parfor-loops. This will enable you to explicitly specify the desired number of threads and work-groups to get the best performance.
- If the global work size (total number of threads to be run in parallel) and the local work size (total number of threads in each work-group) need to be specified explicitly, then the following points should be considered. First, the work-group size (local size) should not be zero, and also it should not exceed the maximum work-group size supported by the parallel device. Second, the local size should be less or equal than the global-size. Third, the

global size should be evenly divisible by the local size.

- The current implementation of OpenCL does not support recursive functions; therefore it is not possible to declare a recursive function as a parallel function.

7 Conclusions

New multi-core CPU and GPU architectures promise high computational power at a low cost if suitable computational algorithms can be developed. The OpenCL C-based parallel programming model provides a way of writing portable parallel algorithms that perform well on a number of multi-core architectures. However, the OpenCL programming model is rather low-level and error-prone to use and intended for parallel programming specialists.

This paper presents the ParModelica algorithmic language extension to the high-level Modelica modeling language together with a prototype implementation in the OpenModelica compiler. This makes it possible for the Modelica modeler to directly write efficient parallel algorithms in Modelica which are automatically compiled to efficient low-level OpenCL code. A benchmark suite called MPAR has been developed to evaluate the prototype. Good speedups have been obtained for large problem sizes of matrix multiplication, Eigen value computation, and stationary heat condition.

Future work includes integration of the ParModelica explicit parallel programming approach with automatic and semi-automatic approaches for compilation of equation-based Modelica models to parallel code. Autotuning could be applied to further increase the performance and automatically adapt it to varying problem configurations. Some of the ParModelica code needed to specify kernel functions could be automatically generated.

8 Acknowledgements

This work has been supported by Serc, by Elliit, by the Swedish Strategic Research Foundation in the EDOp and HIPO projects and by Vinnova in the RTSIM and ITEA2 OPENPROD projects. The Open Source Modelica Consortium supports the OpenModelica work. Thanks to Per Östlund for contributions to Section 3.1.

References

- [1] Modelica Association. *The Modelica Language Specification Version 3.2*, March 24th 2010. <http://www.modelica.org>. Modelica Association.

- Modelica Standard Library 3.1. Aug. 2009. <http://www.modelica.org/>
- [2] Open Source Modelica Consortium. OpenModelica System Documentation Version 1.8.1, April 2012. <http://www.openmodelica.org/>
- [3] Peter Fritzson. *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*. Wiley-IEEE Press, 2004.
- [4] Peter Aronsson. *Automatic Parallelization of Equation-Based Simulation Programs*, PhD thesis, Dissertation No. 1022, Linköping University, 2006.
- [5] Håkan Lundvall. *Automatic Parallelization using Pipelining for Equation-Based Simulation Languages*, Licentiate thesis No. 1381, Linköping University, 2008.
- [6] Håkan Lundvall, Kristian Stavåker, Peter Fritzson, Christoph Kessler: Automatic Parallelization of Simulation Code for Equation-based Models with Software Pipelining and Measurements on Three Platforms. MCC'08 Workshop, Ronneby, Sweden, November 27-28, 2008.
- [7] Per Östlund. *Simulation of Modelica Models on the CUDA Architecture*. Master Thesis. LIU-IDA/LITH-EX-A{09/062}{SE}. Linköping University, 2009.
- [8] Kristian Stavåker, Peter Fritzson. Generation of Simulation Code from Equation-Based Models for Execution on CUDA-Enabled GPUs. MCC'10 Workshop, Gothenburg, Sweden, November 18-19, 2010.
- [9] Matthias Korch and Thomas Rauber. Scalable parallel rk solvers for odes derived by the method of lines. In Harald Kosch, Laszlo Böszörményi, and Hermann Hellwagner, editors, Euro-Par, volume 2790 of Lecture Notes in Computer Science, pages 830-839. Springer, 2003.
- [10] Christoph Kessler and Peter Fritzson. NestStep-Modelica – Mathematical Modeling and Bulk-Synchronous Parallel Simulation. In Proc. of PARA'06, Umeå, June 19-20, 2006. In Lecture Notes of Computer Science (LNCS) Vol 4699, pp 1006-1015, Springer Verlag, 2006.
- [11] Martin Sjölund, Robert Braun, Peter Fritzson and Petter Krus. Towards Efficient Distributed Simulation in Modelica using Transmission Line Modeling. In *Proceedings of the 3rd International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools*, (EOOLT'2010), Published by Linköping University Electronic Press, www.ep.liu.se, In conjunction with MOD-ELS'2010, Oslo, Norway, Oct 3, 2010.
- [12] Francois Cellier and Ernesto Kofman. *Continuous System Simulation*. Springer, 2006.
- [13] Khronos Group, Open Standards for Media Authoring and Acceleration, OpenCL 1.1, accessed Sept 15, 2011. <http://www.khronos.org/opencl/>
- [14] The OpenCL Specication, Version: 1.1, Document Revision: 44, accessed June 30 2011. <http://www.khronos.org/registry/cl/specs/opencl-1.1.pdf>
- [15] NVIDIA CUDA, accessed September 15 2011. http://www.nvidia.com/object/cuda_home_new.html
- [16] NVIDIA CUDA programming guide, accessed 30 June 2011. http://developer.download.nvidia.com/compute/cuda/4_0_rc2/toolkit/docs/CUDA_C_Programming_Guide.pdf
- [17] OpenCL Programming Guide for the CUDA Architecture, Appendix A, accessed June 30 2011. http://developer.download.nvidia.com/compute/DevZone/docs/html/OpenCL/doc/OpenCL_Programming_Guide.pdf
- [18] AMD OpenCL, System Requirements & Driver Compatibility, accessed June 30 2011. <http://developer.amd.com/sdks/AMDAPPSDK/packages/DriverCompatibility.aspx>
- [19] INTEL OpenCL, Technical Requirements, accessed June 30 2011. <http://software.intel.com/enus/articles/opencl-release-notes/>
- [20] OpenCL Work-Item Built-In Functions, accessed June 30 2011. <http://www.khronos.org/registry/cl/sdk/1.0/docs/man/xhtml/workItemFunctions.html>
- [21] Jack J. Dongarra, J. Bunch, Cleve Moler, and G. W. Stewart. LINPACK User's Guide. SIAM, Philadelphia, PA, 1979.
- [22] Ian N. Sneddon. *Fourier Transforms*. Dover Publications, 2010. ISBN-13: 978-0486685229.
- [23] John H. Lienhard IV and John H. Lienhard V. *A Heat Transfer Textbook*. Phlogiston Press Cambridge, Massachusetts, U.S.A, 4th edition, 2011.
- [24] Intel Xeon E5520 CPU Specifications, accessed October 28 2011. [http://ark.intel.com/products/40200/Intel-Xeon-Processor-E5520-\(8M-Cache-2_26-GHz-5_86-GTs-Intel-QPI\)](http://ark.intel.com/products/40200/Intel-Xeon-Processor-E5520-(8M-Cache-2_26-GHz-5_86-GTs-Intel-QPI))
- [25] NVIDIA Tesla M2050 GPU Specifications, accessed June 30 2011. http://www.nvidia.com/docs/IO/43395/BD-05238-001_v03.pdf
- [26] Cyril Faure. *Real-time simulation of physical models toward hardware-in-the-loop validation*. PhD Thesis. University of Paris East, October 2011.

Appendix A. Serial Matrix Multiply

```

model MatrixMultiplication
  parameter Integer m=256 ,n=256 ,k =256;
  Real result ;
algorithm
  result := mainF (m,n,k);
end MatrixMultiplication ;

function mainF
  input Integer m;
  input Integer n;
  input Integer k;
  output Real result ;
protected
  Real A[m,n];
  Real B[n,k];
  Real C[m,k];
algorithm
  // initialize matrix A, and B
  (A,B) := initialize (m,n,k);
  // multiply matrices A and B
  C := matrixMultiply (m,n,k,A,B);
  // only one item is returned to speed up
  // computation
  result := C[m,k];
end mainF;

function initialize
  input Integer m;
  input Integer n;
  input Integer k;
  output Real A[m,n];
  output Real B[n,k];
algorithm
  for i in 1:m loop
    for j in 1:n loop
      A[i,j] := j;
    end for;
  end for;
  for j in 1:n loop
    for h in 1:k loop
      B[j,h] := h;
    end for;
  end for;
end initialize ;

function matrixMultiply
  input Integer m;
  input Integer p;
  input Integer n;
  input Real A[m,p];
  input Real B[p,n];
  output Real C[m,n];
  Real localtmp ;
algorithm
  for i in 1:m loop
    for j in 1:n loop
      localtmp := 0;
      for k in 1:p loop
        localtmp := localtmp + (A[i,k]*
          B[k,j]);
      end for;
      C[i,j] := localtmp ;
    end for;
  end for;
end matrixMultiply;

```

Appendix B. Parallel Matrix-Matrix Multiplication with parfor and Kernel functions

```

model MatrixMultiplicationP
  parameter Integer m=32,n=32,k=32;
  Real result;
algorithm
  result := mainF(m,n,k);
end MatrixMultiplicationP ;

function mainF
  input Integer m;
  input Integer n;
  input Integer k;
  output Real result ;
protected
  Real C[m,k];
  parglobal Real pA[m,n];
  parglobal Real pB[n,k];
  parglobal Real pC[m,k];
  parglobal Integer pm;
  parglobal Integer pn;
  parglobal Integer pk;
  // the total number of global threads
  // executing in parallel in the kernel
  Integer globalSize [2] = {m,k};
  // the total number of local threads
  // in parallel in each workgroup
  Integer localSize [2] = {16 ,16};
algorithm
  // copy from host to device
  pm := m;
  pn := n;
  pk := k;
  (pA ,pB) := initialize(m,n,k,pn ,pk);

  // specify the number of threads and
  // workgroups
  // to be used for a kernel function
  // execution
  oclSetNumThreads(globalSize, localSize);
  pC := matrixMultiply(pn ,pA ,pB );

  // copy matrix from device to host
  // and return result
  C := pC;
  result := C[m,k];

  // set the number of threads to
  // the available number
  // supported by device
  oclSetNumThreads(0);
end mainF ;

function initialize
  input Integer m;
  input Integer n;
  input Integer k;
  input parglobal Integer pn;
  input parglobal Integer pk;
  output parglobal Real pA[m,n];
  output parglobal Real pB[n,k];
algorithm

```

```
parfor i in 1:m loop
  for j in 1: pn loop
    pA[i,j] := j;
  end for;
end parfor;
parfor j in 1:n loop
  for h in 1: pk loop
    pB[j,h] := h;
  end for;
end parfor ;
end initialize ;

parkernel function matrixmultiply
input parglobal Integer pn;
input parglobal Real pA [: ,:];
input parglobal Real pB [: ,:];
```

```
output parglobal Real pC[size(pA,1),
size(pB,2)];
protected
Real plocaltmp ;
Integer i,j;
algorithm
// Returns unique global thread Id value
// for first and second dimension
i := oclGetGlobalId (1);
j := oclGetGlobalId (2);
plocaltmp := 0;
for h in 1: pn loop
  plocaltmp := plocaltmp + (pA[i,h] *
pB[h,j]);
end for;
pC[i,j] := plocaltmp;
end matrixmultiply;
```

Modelling and Simulation of the Coupled Rigid-flexible Multibody Systems in MWorks

Xie Gang¹, Zhao Yan¹, Zhou Fanli*², Chen Liping¹

¹CAD Center, Huazhong University of Science and Technology, Wuhan, China, 430074

²Suzhou Tongyuan Software & Control Tech. Co. Ltd, Suzhou, China, 215123

{xie, zhaoy, zhoufl, chenlp}@tongyuan.cc

Abstract

Aiming to the design challenge of modern mechatronic products, this paper presents a method to simulate the coupled rigid-flexible system in MWorks. Firstly, the component mode synthesis (CMS) technique is introduced and the Craig-Bampton method is adopted to build the flexible-body model. The general flexible-body model named FlexibleBody is developed based on the standard MultiBody library in Modelica, which describes the small and linear deformation behavior (relative to a local reference frame) of a flexible-body that undergoes large and non-linear global motion. In the model, the modal neutral file (MNF) is introduced as a standard interface to describe the constraint modes. Secondly, the model is used to construct a library of boom system of concrete pump truck and the simulations covering the expanding and folding process are carried out based on both the rigid multibody and the coupled rigid-flexible system models. Finally, the influence to dynamics performance of the boom system is analyzed and the conclusion is drawn. The method in this paper provides an effective approach to build unified model and simulate flexible-body in multi-domain engineering systems.

Keywords: rigid-flexible system; concrete pump truck; MWorks

1 Introduction

Much industrial equipment is mechatronic and contains high-speed, lightweight, and high-precision mechanical system. In these mechanical systems one or more structural components often need to consider the deformation effects for design analyses. The integrated design and simulation of the mechatronic systems with flexible bodies make the multidisciplinary challenge. When designing such a mechatronic system, the performance requirements must be satisfied and the strength of the system must be guaran-

teed. Therefore, stress and deformation of machine components have to be predicted in the design process.

The increasing computational power of current computer enables to model a multibody system as 3D deformable body using the finite element method. The flexible multibody dynamics is the subject concerned with the modeling and analysis of constrained deformable bodies that undergo large displacements and rotations. DLR® FlexibleBodies Library^[1,2] provides the general flexible model so that users can simulate the elastic deformation of flexible-body in a modal synthesis way, in which the standard input data (SID)^[3,4] file should be offered by a third-party software. In SID file, Guyan reduction and Ritz approximation are adopted. The Rayleigh-Ritz method^[4] chooses an approximate form for the eigenfunction with the lowest eigenvalue. In the Guyan reduction method^[4], a set of user-defined master nodes are retained and the remaining set of slave nodes are removed by condensation. Only stiffness properties are considered during the condensation, and inertia coupling of master and slave nodes are ignored. Based on an improved Craig-Bampton method^[5-8], MSC.ADAMS® adopts the modal neutral file^[9], which can be exported by some finite element software, to drive the animation of the flexible body. The MNF is a binary file that contains the location of nodes and node connectivity, nodal mass and inertia, mode shapes, generalized mass and stiffness for mode shapes. The mode shapes in modal neutral file, which contain the interface constraint modes, are revised effectively after modal truncation. So the Craig-Bampton method is more accurate and has been widely used in engineering^[10-14]. But the software, just like ADAMS, mainly focuses on modeling and simulation of the pure mechanical system and lacks support of multi-domain physical systems. In order to model and simulate the mechatronic products composed of mechanical, electronic, hydraulic, and control engineering systems, the co-simulation

should be performed with other software such as Matlab/Simulink®, LMS.AMESim®, etc.

In this paper, the FlexibleBody model is developed based on the component mode synthesis (CMS) and the improved Craig-Bampton method^[9, 11]. An external C function MNFParser is programmed to get the mode shapes data in the model. The finite element analysis (FEA) results can then be incorporated into a part model by superimposing the flexible-body deflection on the motion of rigid-body. The postprocessor tool in MWorks^[15] is also improved to support the nephogram animation of the deformation. As an example, a boom system library of the concrete pump truck is developed. And the simulations covering the expanding and folding process are carried out based on both the rigid multibody and the coupled rigid-flexible system models. The simulation results are compared and it shows the coupled rigid-flexible system is more conformable with the actual boom system.

2 The Flexible-Body Model

To build the model of flexible-body based on CMS, the mode shapes data in MNF is provided by third-party finite element analysis (FEM) software. So the MNF file needs to be parsed. Then the equations of coupled elastic deformation and rigid body are established. To animate the deformation of flexible-body, the postprocessor needs to have the ability to show nephograms. The whole process in MWorks is shown in Figure 1.

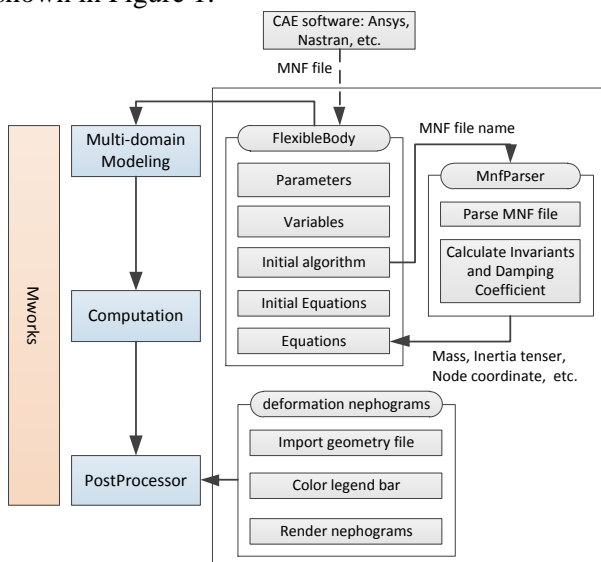


Figure 1: Development process of flexible-body in MWorks

2.1 Theoretical Background

In this paper, only small, linear body deformations relative to a local reference frame are considered, while that local reference frame indicates large, non-linear global motion.

The discretization of a flexible component into a finite element model represents the infinite number of degree of freedom (DOF) with a finite element. The basic premise of modal superposition is that the deformation behavior of a component with a very large number of nodal DOF can be captured with a much smaller number of modal DOF. This reduction in DOF is called modal truncation.

The linear deformations of the nodes of this finite element mode, \mathbf{u} , can be approximated as a linear combination of a smaller number of translational mode shapes matrix^[9], $\Phi_T = (\phi_{T1}, \phi_{T2}, \dots, \phi_{TM})$.

$$\mathbf{u} = \Phi_T \mathbf{q} + \bar{\Phi}_T \bar{\mathbf{q}} \approx \phi_{T1} q_1 + \dots + \phi_{TM} q_M = \sum_{i=1}^M \phi_{Ti} q_i \quad (1)$$

Where $\bar{\Phi}_T$ and $\bar{\mathbf{q}}$ are the truncated mode shapes matrix and the truncated modal coordinates respectively. M is the number of mode shape, and $\mathbf{q} = (q_1, q_2, \dots, q_M)^T$ is the modal coordinates.

Similar to equation(1), as the body deforms, every node rotates through small angles relative to its reference frame. These angles are obtained using a modal superposition^[9].

$$\boldsymbol{\theta} = \Phi_R \mathbf{q} + \bar{\Phi}_R \bar{\mathbf{q}} \approx \sum_{i=1}^M \phi_{Ri} q_i \quad (2)$$

Where $\Phi_R = (\phi_{R1}, \phi_{R2}, \dots, \phi_{RM})$ is the slice of rotational mode shapes matrix.

In the studies referring to [3, 4], the eigenvectors of an unconstrained system be used. Eigenvectors are found to provide an inadequate basis in system level modeling. To improve the accuracy of the system model, the CMS^[3] techniques are adopted, the most general methodology is Craig-Bampton method^[5].

The Craig-Bampton method^[5, 6] allows the user to select a subset of DOF which are not to be subject to modal superposition. These boundaries DOFs (or attachment DOFs) are preserved exactly in the Craig-Bampton modal basis. There is no loss in resolution of these DOF when higher order modes are truncated^[7, 8].

The system DOF in Craig-Bampton method are partitioned into boundary DOF, \mathbf{u}_b , and interior DOF, \mathbf{u}_I . Two sets of mode shapes are defined, as follows^[9]:

Constraint modes: These modes are static shapes obtained by giving each boundary DOF a unit displacement while holding all other boundary DOF

fixed. The basis of constraint modes completely spans all possible motions of the boundary DOFs, with a one-to-one correspondence between the modal coordinates of the constraint modes and the displacement in the corresponding boundary DOF, $\mathbf{q}_C = \mathbf{u}_B$.

Fixed-boundary normal modes: These modes are obtained by fixing the boundary DOF and computing an eigensolution. There are as many fixed-boundary normal modes as the user desires. These modes define the modal expansion of the interior DOF. The quality of this modal expansion is proportional to the number of modes retained by the user.

The relationship between the physical DOF and the Craig-Bampton modes and their modal coordinates is illustrated by the following equation.

$$\mathbf{u} = \begin{Bmatrix} \mathbf{u}_B \\ \mathbf{u}_I \end{Bmatrix} = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \Phi_{IC} & \Phi_{IN} \end{bmatrix} \begin{Bmatrix} \mathbf{q}_C \\ \mathbf{q}_N \end{Bmatrix} \quad (3)$$

Where \mathbf{I} , $\mathbf{0}$ are identity and zeros matrices, respectively. Φ_{IC} is the physical displacements of the interior DOF in the constraint modes. Φ_{IN} is the physical displacements of the interior DOF in the normal modes. \mathbf{q}_C is the modal coordinates of the constraint modes. \mathbf{q}_N is the modal coordinates of the fixed-boundary normal modes.

The generalized stiffness and mass matrices corresponding to the Craig-Bampton modal basis are obtained via a modal transformation.

2.2 FlexibleBody Model

The governing differential equation of flexible-body^[9, 15], in terms of the generalized coordinates is:

$$\mathbf{M}\ddot{\xi} + \mathbf{M}\dot{\xi} - \frac{1}{2} \left[\frac{\partial \mathbf{M}}{\partial \xi} \dot{\xi} \right]^T \dot{\xi} + \mathbf{K}\xi + \mathbf{f}_g + \mathbf{D}\dot{\xi} + \left[\frac{\partial \Psi}{\partial \xi} \right]^T \lambda = \mathbf{Q} \quad (4)$$

Where,

$\xi, \dot{\xi}, \ddot{\xi}$ are the generalized coordinates of the flexible-body and their time derivatives.

$$\xi = (x \ y \ z \ \psi \ \phi \ \phi \ q_{i,(i=1,\dots,M)})^T = (\mathbf{x} \ \Psi \ \mathbf{q})^T$$

\mathbf{M} is the mass matrix.

\mathbf{K} is the generalized stiffness matrix.

\mathbf{f}_g is the generalized gravitational force.

\mathbf{D} is the modal damping matrix.

Ψ is the algebraic constraint equations.

λ is the Lagrange multipliers for the constraints.

\mathbf{Q} is the generalized applied forces.

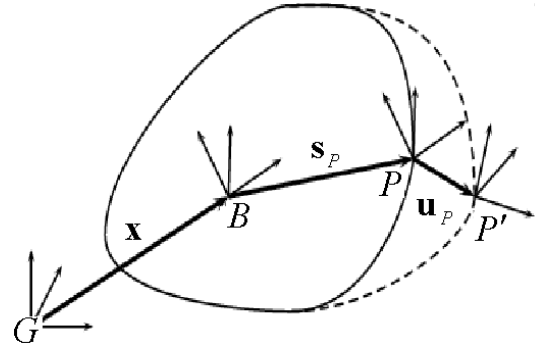


Figure 2: The position vector to a deformed point P' on a flexible body

The instantaneous location of a point that is attached to a node, P , on a flexible body, B , is the sum of three vectors, showing in Figure 2.

$$\mathbf{r}_p = \mathbf{x} + \mathbf{s}_p + \mathbf{u}_p \quad (5)$$

Where

\mathbf{x} is the position vector from the origin of the ground reference frame to the origin of the local body reference frame of the flexible body.

\mathbf{s}_p is the position vector of the undeformed position of point P with respect to the local body reference frame of body B .

\mathbf{u}_p is the translational deformation vector of point P , the position vector from the point's undeformed position to its deformed position. It is also expressed in the local body coordinate system. The deformation vector is a modal superposition, $\mathbf{u}_p = \Phi_{TP}\mathbf{q}$. Where Φ_{TP} is the slice from the modal matrix that corresponds to the translational DOF of node P .

The general flexible-body model based on CMS is developed according to the following processes.

- (1) Defining the parameters: MNF file name, mass, inertia, mode shape, set of selected mode, etc.
- (2) Setting the variants and default values: modal coordinates and first-order derivate, second-order derivate, velocity, acceleration, etc.
- (3) Configuring the initial algorithm: Call MNFParser function (refer to 2.3 MNF File Parser) to get the mass, inertia, mode shape, stiffness matrix, invariants, damping coefficient, etc.
- (4) Setting the initial equations: just like the equations in Body model in Multibody library.
- (5) Describing the equations: Force and torque balance equation, and equation (4) are defined. With modal coordinates, the deformation of the flexible-body equation (1), (2), (4) and (5).

This approach consists of the Body model in Multibody library. The general FlexibleBody model is shown in Figure 3.

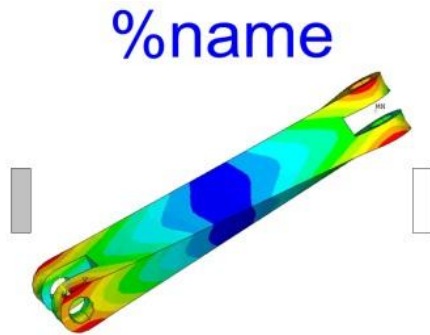


Figure 3: Icon of the FlexibleBody model

The FlexibleBody model encapsulates the complexity of details into a black box that we can use modularly without considering the detailed implementation at the top level.

2.3 MNF File Parser

The mode shapes data are needed to build the flexible-body model with modal synthesis method, and the MNF is adopted to express these data, which is generated by CAE software such as ANSYS®, NASTRAN® etc.

Modal neutral file is a platform-independent binary file. The information in a MNF includes geometry (locations of nodes and node connectivity), nodal mass and inertia, mode shapes, generalized mass and stiffness for mode shapes, which is listed in Table 1.

Table 1: Information in MNF

Block	information
Header	date, program name and version, title, MNF version, units
Body properties	mass, moments of inertia, center of mass
Interface points	Reduced stiffness and mass matrices
Interface modes	User requests the number of modes
Constraint modes	Interface constraint modes

According to its data structure, an external C function “MNFParser” is programmed to parse the MNF file and to obtain the quality, inertia tensor, eigenvalues, modal shapes matrix, etc.

To reduce the simulation time, the nine inertia invariants^[9] are calculated beforehand from the N nodes of the finite element model based on each node's mass, undeformed location coordinates in the component modes.

And the default damping coefficients are calculated according to the modal frequency.

- 1% damping for all modes with frequency lower than 100.
- 10% damping for modes with frequency in the 100-1000 range.

- 100% critical damping for modes with frequency above 1000.

The MNFParser also provides nodes coordinates and element faces of the FE model to the postprocessor for rendering deformation nephograms.

3 Modeling and Simulation of the Coupled Rigid-flexible Boom System

The concrete pump truck has become a kind of indispensable machinery equipment in construction industry. It pumps concrete continuously sent by concrete mixer truck to the pouring site. The boom system is generally composed of mechanical, hydraulic and control subsystems, and the multi-domain modeling and simulation is necessary for the design and validation of this system. The modeling and simulation process of boom system in MWorks is shown in Figure 4.

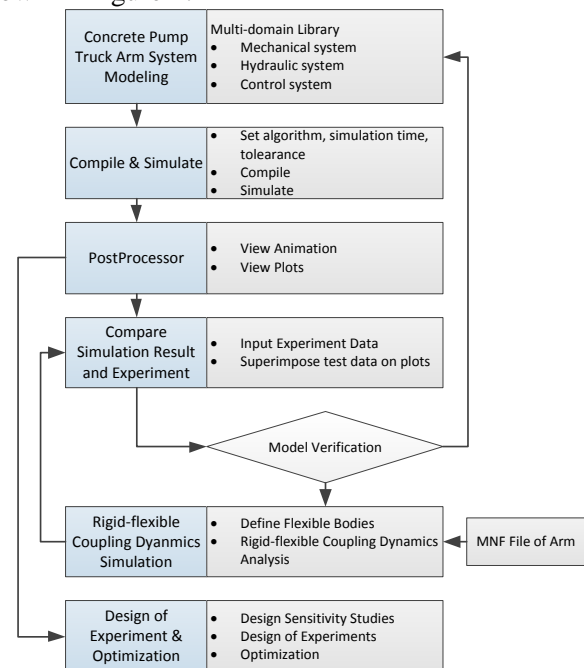


Figure 4: The modeling and simulation workflow of booms system of concrete pump truck

3.1 Library of Boom System

The hierarchical multi-domain library of boom system is developed based on the standard Modelica library and the hydraulic library developed by Suzhou Tongyuan Software & Control Technology Company. Its structure is shown in Figure 5.

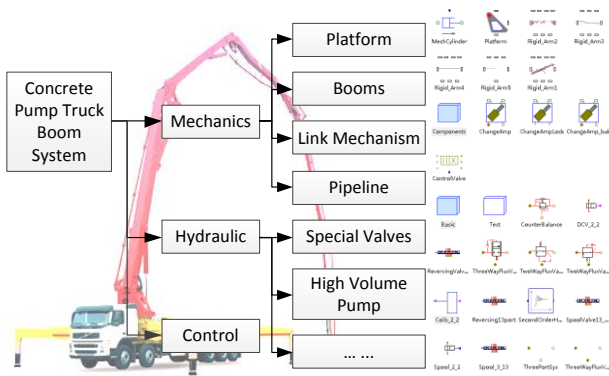


Figure 5: Structure of the booms library

The boom system library includes mechanics, hydraulic, and control subsystem library. The boom

system model can be conveniently constructed by dragging and dropping based on the library.

3.2 Simulation of Rigid Boom System

The booms of concrete pump truck must expand and fold regularly while working. The working loads should be analyzed to ensure the safety. The control system should also be designed to satisfy the casting needs.

The hierarchical structure of the boom system model is established based on the MultiBody library, shown in Figure 6.

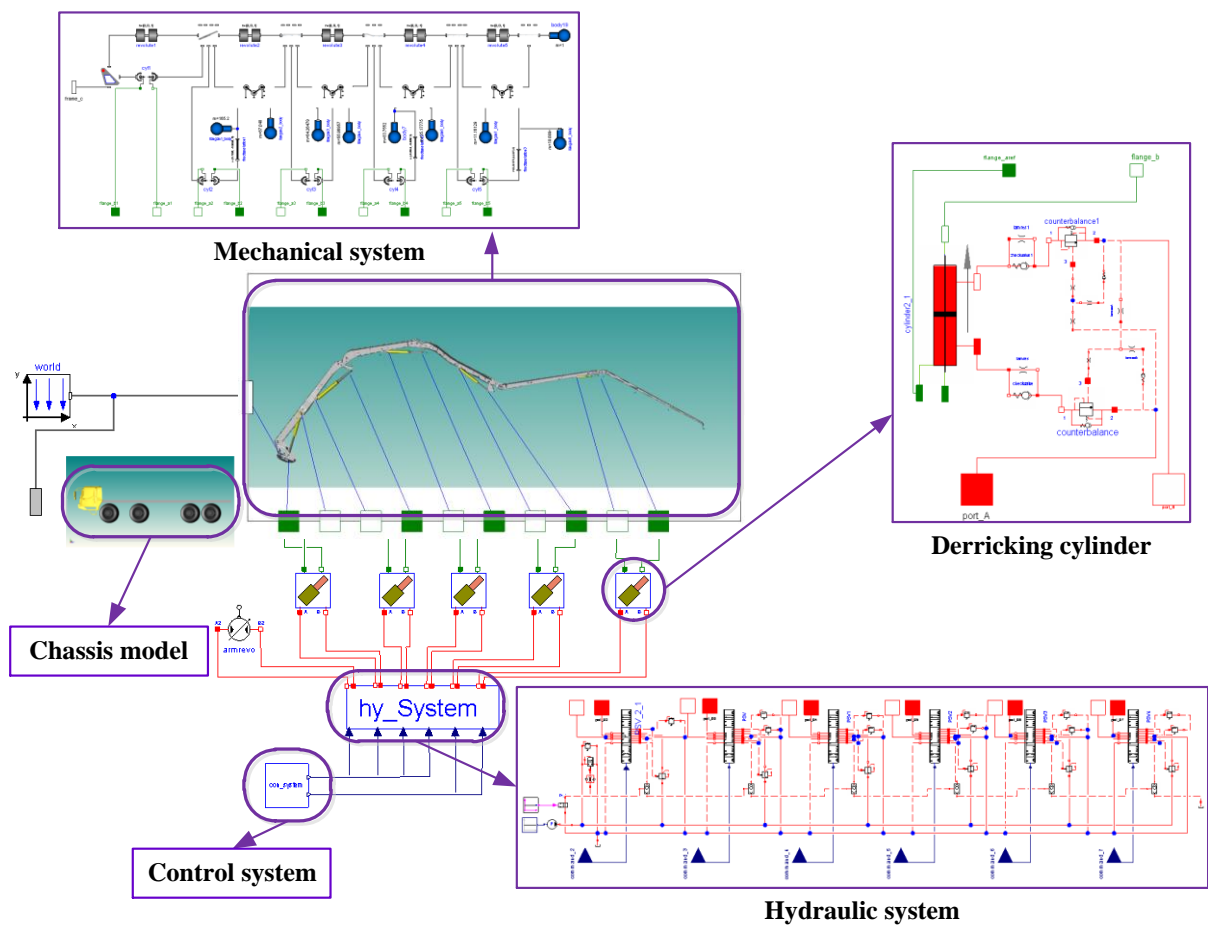


Figure 6: Hierarchical model of the booms system

Base on the system model, the various working conditions can be simulated by modifying the parameters of the control and hydraulic systems, and the work loads of each boom, flow quantity and pressure, impact loads in the hydraulic system and the reliabil-

ity of control system can also be analyzed. For example, the expanding and folding process of the boom₄ is simulated and the result is shown in Figure 7.

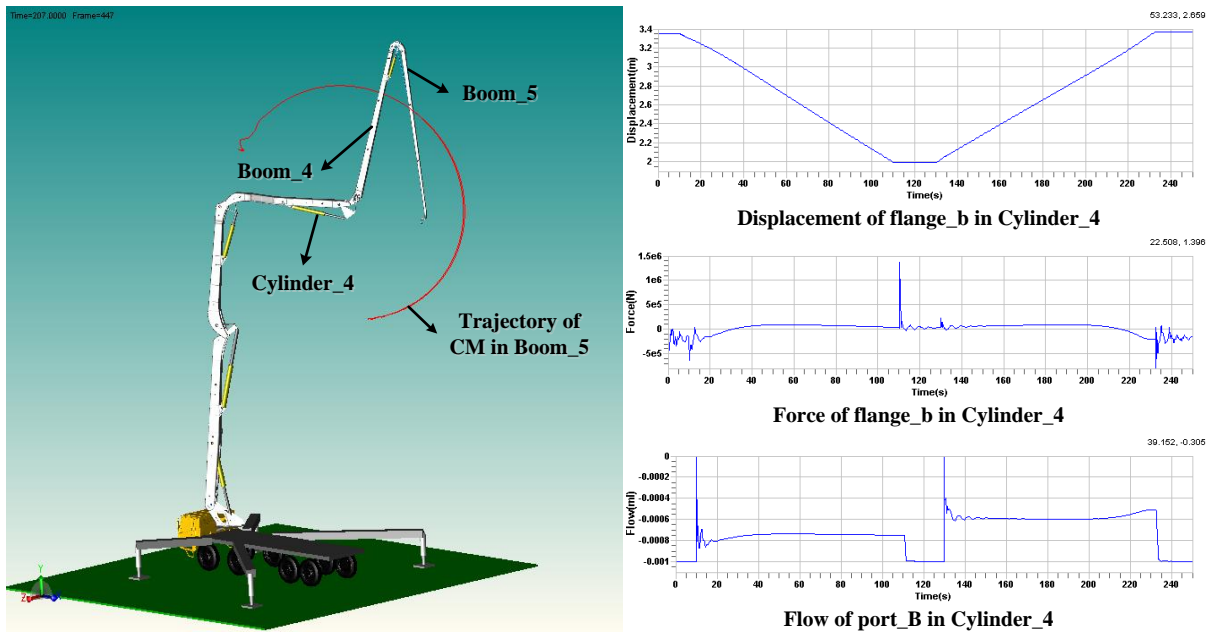


Figure 7: The expanding and holding process of the boom_4

3.3 Simulation of the Coupled Rigid-flexible Boom System

There will be a large elastic deformation on each boom in the actual working process. This not only has a great impact on the casting work, but also reduces the safety performance of the system. So it is necessary to take the elastic deformation into consideration for improving the accuracy of the system model.

3.3.1 Modal Analysis of the Booms

The modal neutral files of booms are needed to perform the coupled rigid-flexible dynamic analysis. So it's necessary to take modal analysis for each boom. We use the ANSYS software to compute the modals. The analysis workflow is as follow:

- (1) Inputting the material parameters: The elastic modulus, Poisson's ratio, and density are set to FE model. The material properties determine the spring stiffness and damping.
- (2) Meshing the model: The solid45 element is selected to mesh the solid geometry. And the mass21 element is selected to mesh the key points set up on the central axis of the hinge hole.
- (3) Configuring the rigid region: The rigid regions are established about the interface node and nodes on the cylinder faces respectively.
- (4) Generating the MNF: Run the ADAMS macros, choose the interface nodes, specify the mode order numbers to expand, then generate modal neutral file of each boom.

The finite element models of booms are shown in Figure 8.

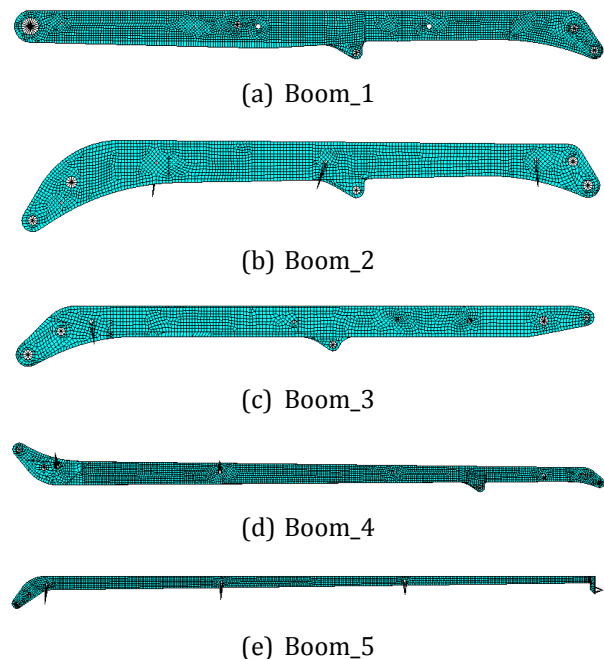


Figure 8: FE model of booms

We set twenty modes to extract for each boom. Then we can get fifty modes in the MNF. The modes from first to sixth are rigid modes, approximately to zero. Because there are five fixed interface points in each boom, and thirty constraint modes of six degrees of freedom are extracted. We choose a sufficient number of modes to represent the boom so that the frequency range is able to deactivate eigenmodes based on the frequency or the energy criterion. Some natural frequencies of the booms are listed in Table 2.

Table 2: Natural frequency of booms

Nat. Freq.	Boom_1	Boom_2	Boom_3	Boom_4	Boom4
1-6	0	0	0	0	0
7	25.8302	35.5842	33.0956	49.3527	8.7492
8	36.3539	48.4689	49.3527	22.1214	11.1239
9	52.8219	76.9397	49.3527	53.6334	23.7672
10	74.1143	89.2075	49.3527	60.5197	30.3335
11	75.6471	90.2372	49.3527	81.8889	47.3777
12	81.2592	94.7992	49.3527	85.3635	56.8012
...

mechanical system is converted from the rigid multi-body to the coupled rigid-flexible system.

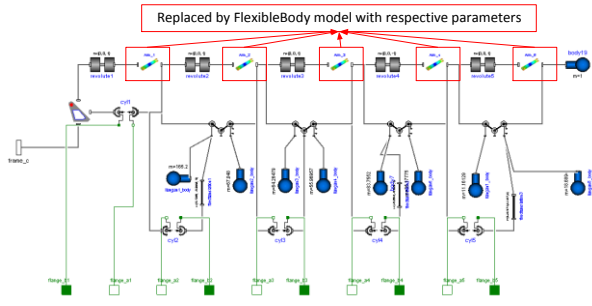


Figure 9: The coupled rigid-flexible mechanical system

3.3.2 Replacement of the FlexibleBody Model

The rigid parts of booms shown in Figure 6 are replaced by FlexibleBody model with respective parameters, as shown in Figure 9. So the model of the

3.3.3 Simulation of Coupled Rigid-flexible Boom System

The expanding and folding process of boom_5 is simulated, as shown in Figure 10.

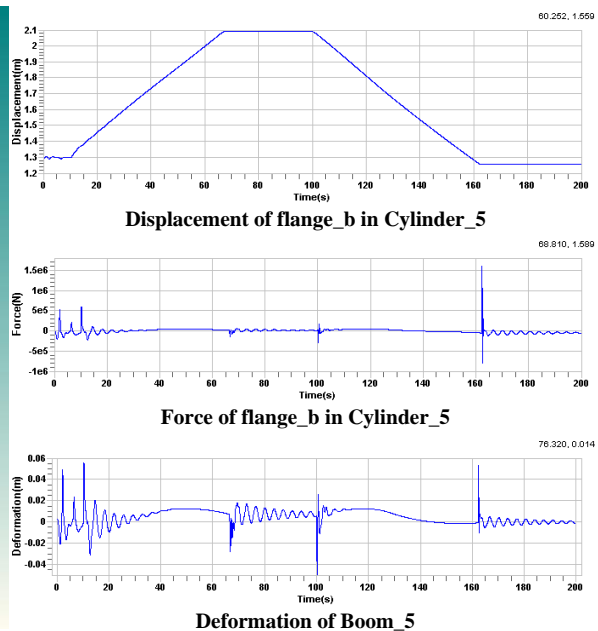
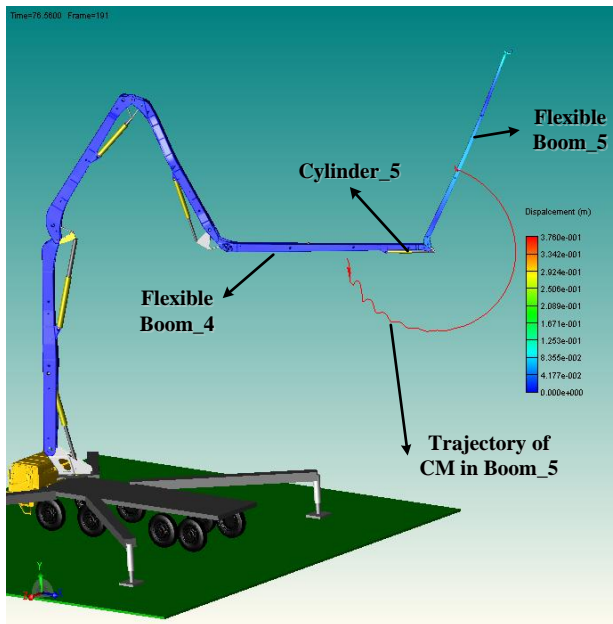
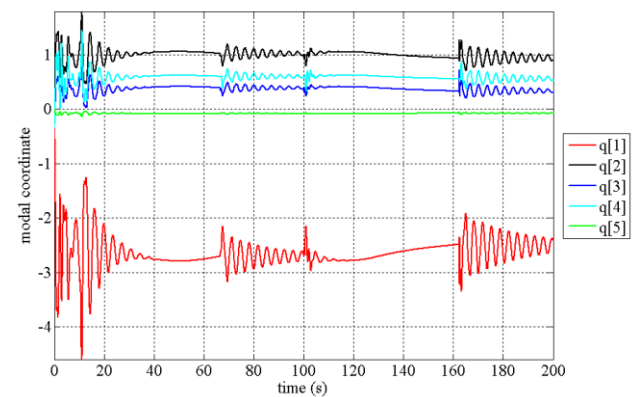


Figure 10: The process of expanding and folding boom_5

The modal coordinates of each boom are shown in Figure 11. The modal coordinates $q[1] - q[6]$ are corresponding to 7th - 12th modes respectively. Obviously, the value of modal coordinate $q[1]$ is the biggest one in each boom. It indicates that the 7th mode contributes most energy to the flexible-body. And the values of other modal coordinates are smaller and smaller, indicating less energy contribution. The variation tendency is complied with the modal superposition theorem and energy criterion^[11].



(a) Boom 1

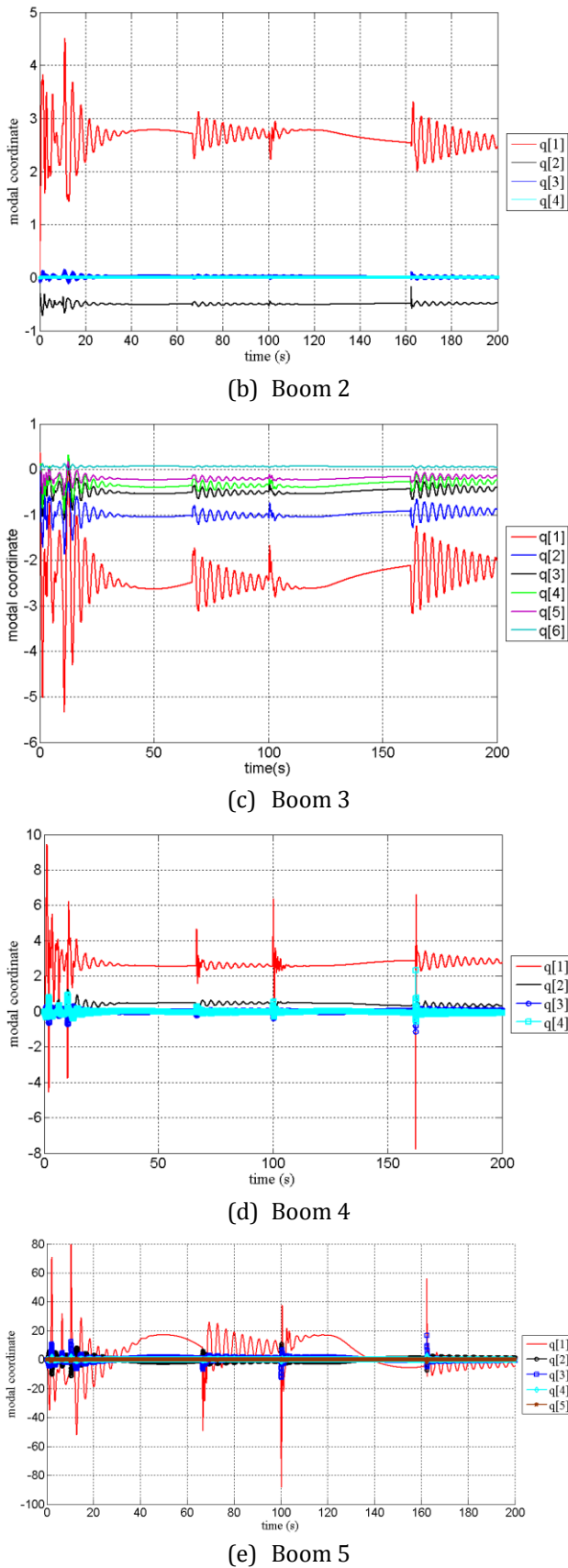


Figure 11: Modal coordinates of booms

With the powerful post-processor in MWorks, the deformation color contour of boom_4 and boom_5 are shown in Figure 12.

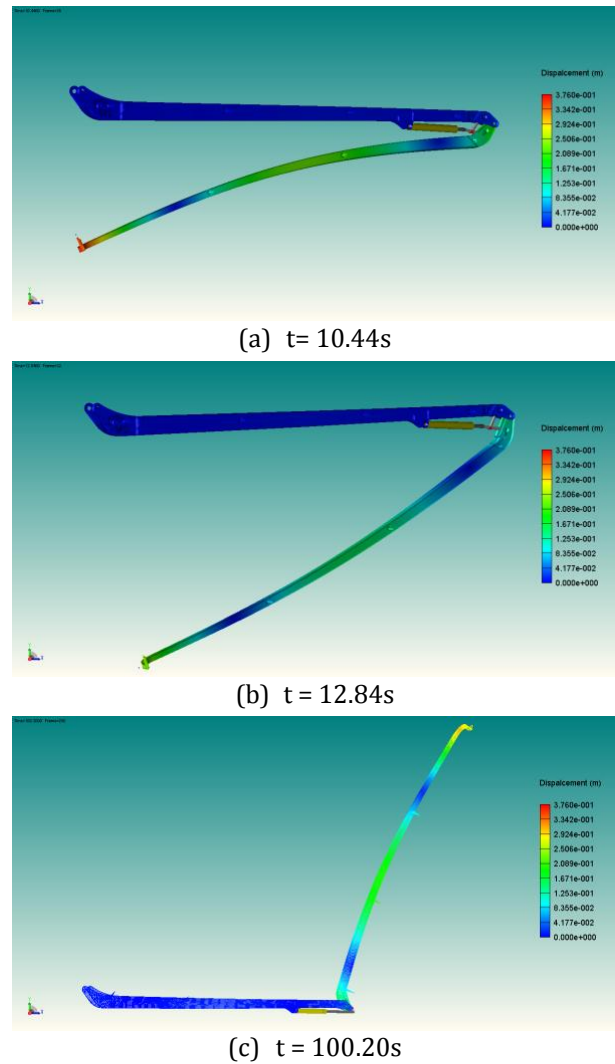


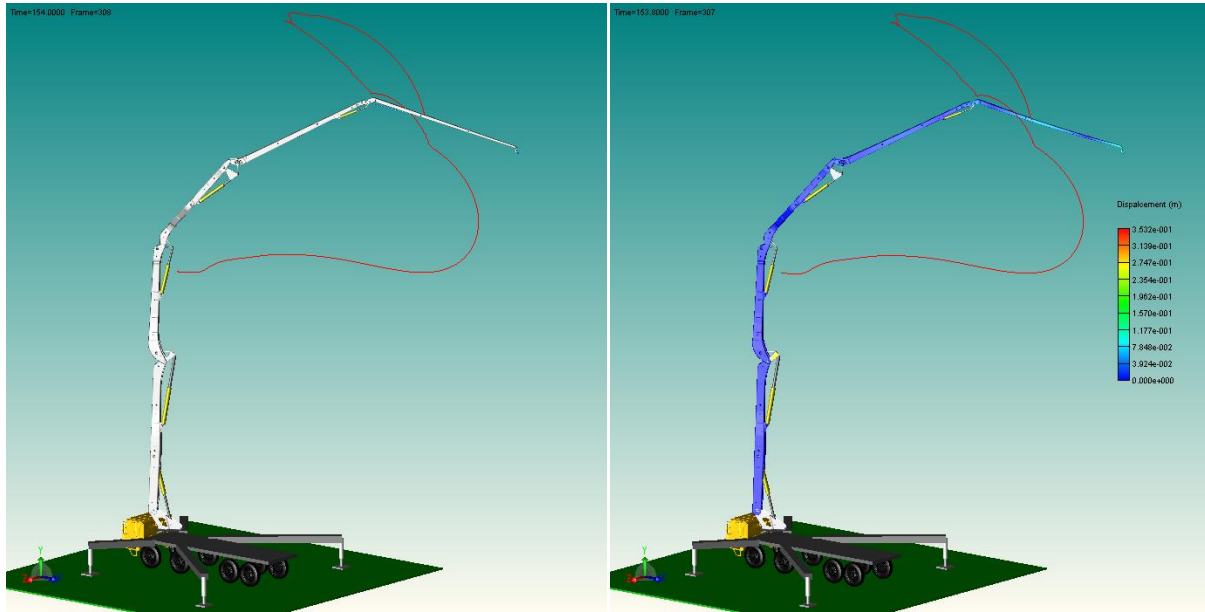
Figure 12: Nephograms of the deformation in booms

The obvious vibration can be found in simulation result of the coupled rigid-flexible dynamics. And in the actual working process, the vibration does exist in the expanding or the holding process. The impact on hydraulic and control system can be analyzed for the elastic deformation of each boom. So compared with a rigid model, the coupled rigid-flexible model has higher accuracy, and is more close to the actual system.

3.4 Simulation Comparison between Rigid and Coupled Rigid-flexible Boom System

3.4.1 Comparison of Simulation Results

The Figure 13 shows the simulation results, in comparison coupled rigid-flexible multibody with rigid multibody, of boom_3 ~ boom_5 expanding and holding together.



Rigid boom system

Coupled rigid-flexible boom system

Figure 13: Comparison of rigid system and coupled rigid-flexible system

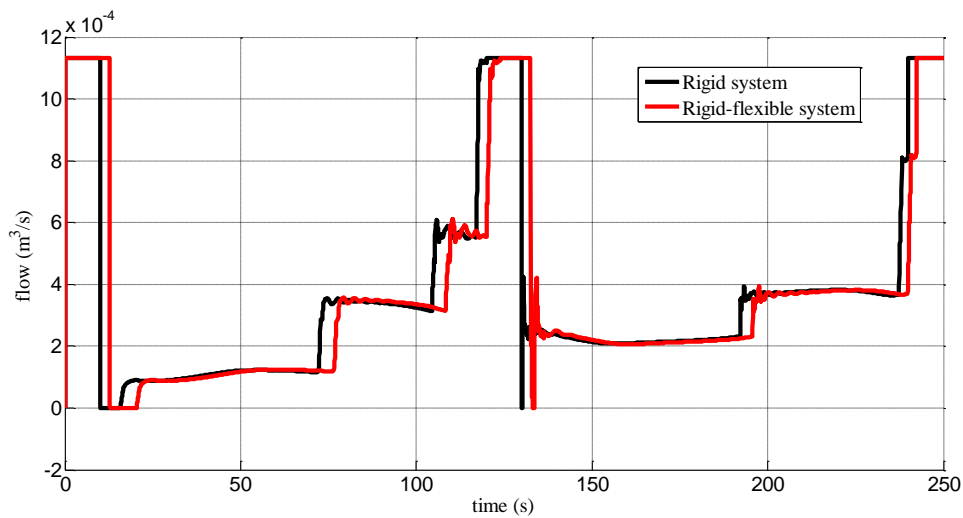


Figure 14: Flow in hydraulic cylinder port_A of boom_4

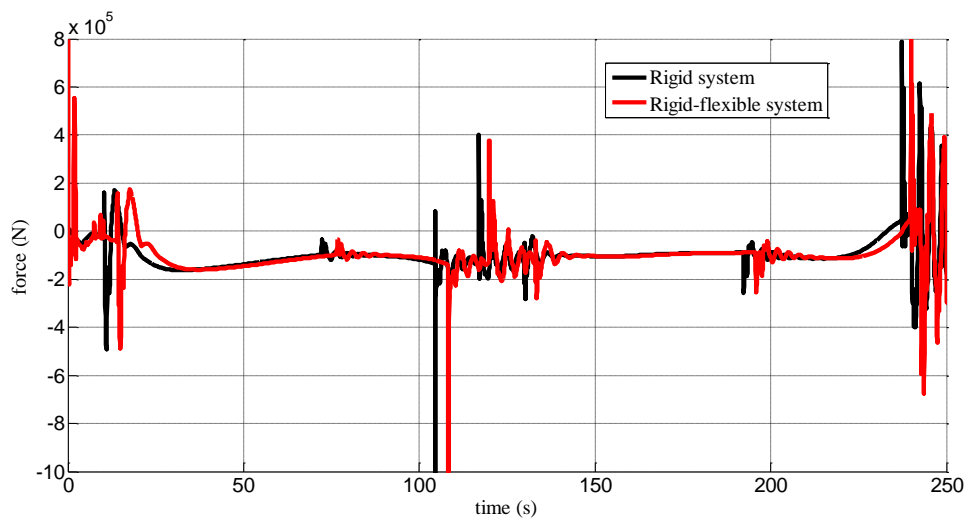


Figure 15: Force in flang_b of boom_4

Figure 14 shows the flow in hydraulic cylinder port_A of boom_4. Figure 15 shows the force in flange_b of boom_4. There are obvious differences between rigid and the coupled rigid-flexible boom

system, especially at the reversing point: the former changes gently, but the latter changes dramatically, which are due to elastic deformation of flexible booms.

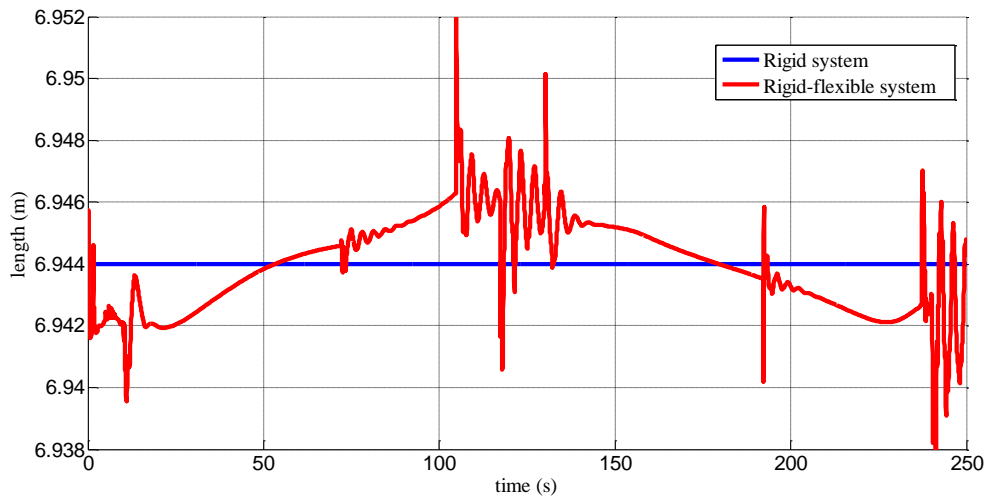


Figure 16: Length of boom_4

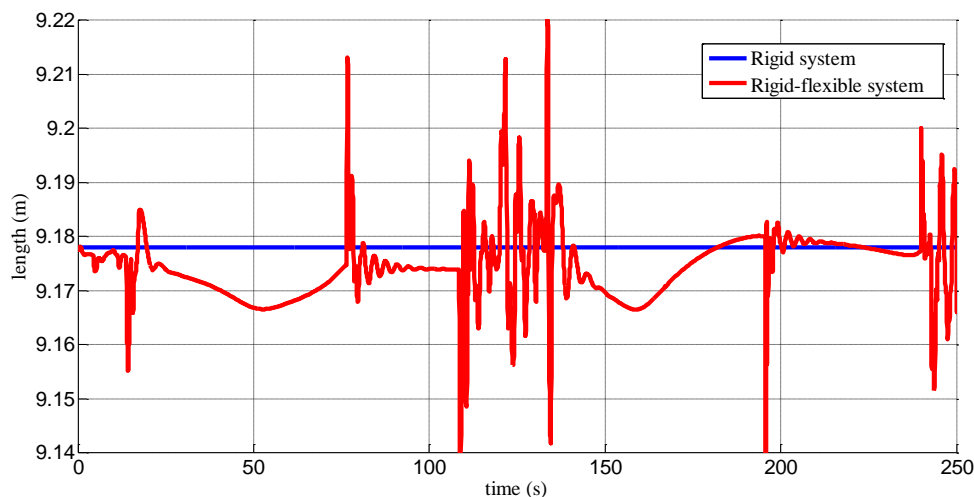


Figure 17: Length of boom_5

Figure 16 and Figure 17 show the length of boom_4 and boom_5 respectively. The length of the rigid boom is a constant value, while the length curve of the flexible boom is shown a relatively evident fluctuation, especially when the motion state changed, deformation peak appears. The maximum force of each hinge point is larger than the rigid body simulation results, when boom system changes the motion state.

3.4.2 Comparison of Calculation Efficiency

The finite element method is apt for discretizing the arbitrary complicated geometry. But with gigantic nodes in large scale and complex system, the computation is heavy. FEA is too inefficient for system level modeling and is incapable of analyzing large mo-

tion. Moreover, the coupled rigid-flexible system is a strong nonlinear system, especially integrated with hydraulic and control subsystem.

Although introducing great number of variables and equations, component mode synthesis saves time and processing resources by breaking up a single large problem several reduced-order problems via substructuring.

The calculation time for coupled rigid-flexible boom system is more than double for rigid boom system in the test cases. There are several influence factors for calculation time.

- (1) The number of nodes. Too many DOFs can mean unacceptably long computation time.
- (2) The number of modes. If a mode does not contribute to the response of the flexible component

during a simulation, it could be disabled to save computation time.

- (3) The modal damping coefficients. The bigger modal damping coefficient is helpful to control the integration step by suppressing the resonance response respect to the corresponding natural frequency.

4 Conclusions

By introducing the CMS technique and the improved Craig-Bampton method, the FlexibleBody model has been constructed based on the standard MultiBody library in this paper. The boom system of concrete pump truck is modeled and simulated in MWorks, which is composed of a coupled rigid-flexible mechanism, a hydraulic and a control subsystem. Numerical results are compared and discussed with respect to efficiency and accuracy.

The FlexibleBody model can be easily incorporate flexibility into system models. This optional add-on module interface with several commercial finite element applications to accurately define component flexibility, and it has an easy-to-use interface that allows engineers to quickly convert rigid parts to flexible ones.

The simulation process is illustrated by boom system of concrete pump truck applications. However, it can be applied to design any mechanical system such as classical or compliant mechanisms, deformable structures and more general to solve most mechanical dynamics problems.

ACKNOWLEDGEMENTS

The paper is supported by Major National Science & Technology Specific Project (No. 2011ZX02403-005), The National Basic Research Program of China (973 Program) (No. 2011CB706502).

References

- [1] Andreas Heckmann, Martin Otter, Stefan Dietz, et al. The DLR FlexibleBodies library to model large motions of beams and of flexible bodies exported from finite element programs, The Modelica Association. Modelica 2006: 85-95.
- [2] Andreas Heckmann, Stefan Hartweg and Ingo Kaiser. An Annular Plate Model in Arbitrary Lagrangian-Eulerian Description for the DLR FlexibleBodies Library. Proceedings 8th Modelica Conference, Dresden, Germany, March 20-22, 2011: 121- 132.
- [3] O. Wallrapp: Standardization of Flexible Body Modeling in Multibody System Codes, Part 1: Definition of Standard Input Data, Mechanics of Structures and Machines 22(3):283-304, 1994.
- [4] P. Koutsovasilis, V. Quarz, M. Beitelshmidt. Standard input data for FEM-MBS coupling: importing alternative model reduction methods into SIMPACK. Mathematical and Computer Modelling of Dynamical Systems, 2009, 15(1): 51-68.
- [5] R. Craig, M. Bampton. Coupling of substructures for dynamic analysis. Amer. Inst. Aero. Astro. J. 1968, 6(7): 1313-1319.
- [6] S. Rubin. Improved component-mode representation for structural dynamic analysis Amer. Inst. Aero. Astro. J., 13 (8) (1975), pp. 995-1006.
- [7] R. R. Craig. Coupling of substructures for dynamic analyses: an overview, in: Structures, Structural Dynamics and Material Conference, 41st AIAA/ASME/ASCE/AHS/ASC, Atlanta, 2000, AIAA-2000-1573.
- [8] D. J. Daniel, J. Rixen. A dual Craig-Bampton method for dynamic substructuring. Journal of Computational and Applied Mathematics, 2004, 168(1-2): 383-391.
- [9] ADAMS. Theoretical Background. MSC. Software Corporation, 2003: 1-30. http://ti.mb.fh-osna-brueck.de/adamshelp/mergedProjects/flex/flex_gen/flextheory.pdf.
- [10] Dimitri Metaxas, Eunyoung Koh. Flexible multibody dynamics and adaptive finite element techniques for model synthesis and estimation. Comput. Methods Appl. Mech. Engrg. 1996, 136: 1 – 25.
- [11] Takehiko Eguchi. Improvement of component mode synthesis model for vibration analysis of hard disk drives using attachment modes. Microsyst technol, 2007, 13: 1085-1092.
- [12] Frédéric Bourquin. Analysis and comparison of several component mode synthesis methods on one-dimensional domains. Numer. Math. 1990, 58: 11 – 34.
- [13] Polarit Apiwattanalungarn, Steven W. Shaw, Christophe Pierre. Component Mode Synthe-

- sis Using Nonlinear Normal Modes. *Nonlinear Dynamics*, 2005, 41: 17 – 46.
- [14] Ulf Sellgren. Component Mode Synthesis – A method for efficient dynamic simulation of complex technical systems. Department of Machine Design, the Royal Institute of Technology, Sweden, 2003: 1-27.
- [15] Zhou Fanli, Chen Liping, Wu Yizhong, et al. MWorks: a Modern IDE for Modeling and Simulation of Multi-domain Physical Systems Based on Modelica. *The Modelica Association, Modelica 2006*: 725-731.

A Modelica Library of Anisotropic Flexible Beam Structures for the Simulation of Composite Rotor Blades

Christian Spieß Manfred Hajek
Technische Universität München, Institute for Helicopter Technology
Boltzmannstr. 15, 85748 Garching

Abstract

Beam theories are extensively used for simulation of helicopter rotor blades. The predominant deployment of composite materials in rotor blade development demands for complex theories that are able to describe the elastic behavior of anisotropic and nonhomogeneous materials. In this paper a Modelica library is presented which is capable of simulating extensional, torsional and flexural deformation and the couplings between those degrees of freedom. The structural dynamic model is based on cross-sectional analysis.

Keywords: flexible structures; rotating beams; helicopter rotor design; cross-sectional analysis

1 Introduction

The non-linear static and dynamic analysis of bent and twisted beams is of major importance for many engineering disciplines. Especially for helicopter rotor applications beam models are used to simulate its dynamic behavior. Since helicopter rotor blades are made of composite structures and materials that may be anisotropic or nonhomogeneous the long and slender beam structure is subject to non-classical effects such as transverse shear deformation, geometric nonlinearities, cross-sectional warping, and elastic coupling [4]. Thus classical beam theories limited to isotropic materials and simple cross-sectional geometries may not be applicable in rotorcraft analysis codes. Some vibration phenomena in particular with significant bending-torsion coupling require adequate modeling. Therefore a sophisticated beam theory has been implemented to Modelica which has extensively been tested in practical applications such as the CAMRAD II [7] rotorcraft analysis code and been proven to provide satisfactorily results [8]. One of the key features of modern beam theories is the cross-sectional analysis. It splits the problem into a two-dimensional analysis of the cross-section and the one-dimensional beam

kinematics. The two-dimensional analysis provides the structural and inertial parameters that can be set at any number of points on the beam. Hence the influence of anisotropy and inhomogeneity can be taken into account and these methods are not limited to specific materials. The one-dimensional analysis provides the elastic equations of motion to calculate the kinematics of the beam. The main advantage of this approach as opposed to three-dimensional finite element analysis is the significant reduction of computational effort. It allows the calculation of elastic beam behavior in multi-body environments as well as real time applications.

2 Modeling Capabilities

The flexible beam library is a Modelica package to model elastic motion of beam-like structures represented by axial, bending, and torsion deflection of a beam with arbitrary cross-sectional geometry. To attach the beam model to the simulation environment the standard Modelica frame connector from the Multi-Body library is used. In addition to the connectors at each end of the beam an arbitrary number of frames on the beam axis may be defined to connect to other system components such as joints, sensors, or force elements. Figure 1 shows an exemplary setup with two beam segments connected in series. The user has the ability to define an arbitrary number of cross-sectional properties along the beam axis including the stiffness parameters and inertial properties. Those parameters can be obtained by NABSA [5] or VABS [1], which are both 2D cross-sectional analysis tools for general nonhomogeneous and anisotropic beam sections including warp and twist.

The library features two options to model the beam:

1. An Euler-Bernoulli beam theory for isotropic beam materials with St. Venant torsion.

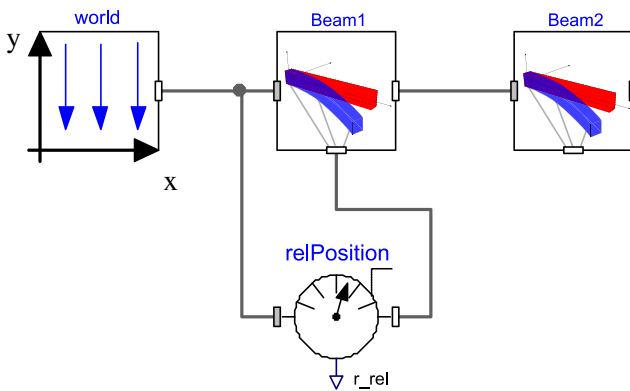


Figure 1: example beam setup

2. A beam theory for anisotropic or composite materials including transverse shear deformation.

The effects of cross-section warping and transverse shear are included in the section structural properties. Their effect on inertial forces and interface geometry is neglected, warp variables are expressed in terms of strain measures. The undeflected beam axis is assumed to be straight within the component. Thus a beam geometry defined by initial curvature or kinks must be split into several straight beam components. Also the theory requires the integration of beam properties along the beam axis which is implemented as Gaussian integration. Hence structural and inertial properties must not vary rapidly along its length.

3 Beam Theory

The motion of the beam element is represented by the rigid body motion of a reference frame at one end of the beam and an elastic motion relative to this frame. Due to this decomposition it is possible to superimpose an arbitrary large motion of the reference frame, which will be treated in a correct manner, by a small elastic motion approximated to the second order. The representation of the elastic deformation is based on references [7, 6]. The reference frame defines the coordinate system B at the origin of the undeflected beam axis. This axis extends on the positive x -axis of system B from $x = 0$ to $x = l$. The elastic motion of a point on the beam axis is described by four parameters; namely the constant position x on the undeflected axis plus the elastic elongation u as well as the bending deflections v and w that cause a rotation of the cross-section. Additionally the parameter θ is used to describe the rotation around the beam axis (see Fig. 2). The tangent of the cross-section at this point is rotated by the angles β and ζ around the y and z axes respec-

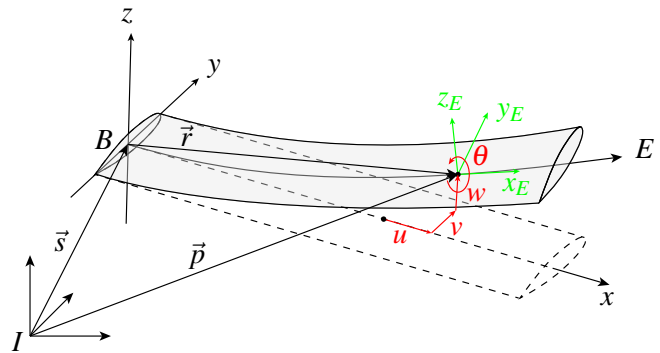


Figure 2: motion of the beam

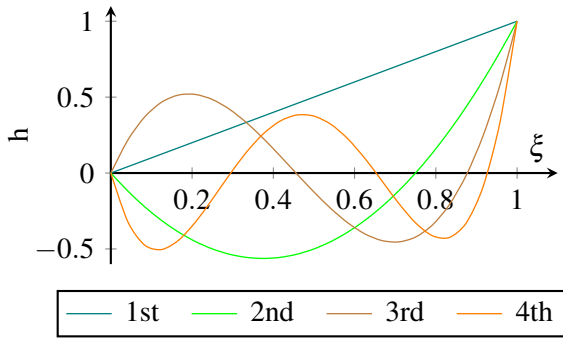
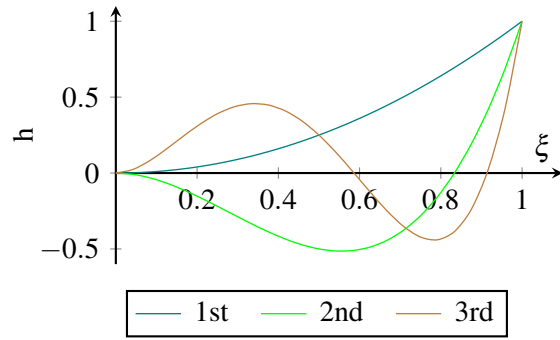
tively produced by bending deflection, which can be obtained by $\sin \beta = w'$ and $\sin \zeta = v'$ for second order approximation¹. That leads to the transformation matrix $C^{EB}(x)$ between the origin of the beam and the bent and twisted axes of the coordinate system E at position x of the beam axis. Euler angles are used to describe the rotation $C^{EB} = X_{\theta}Y_{-\beta}Z_{\zeta}$, where X, Y , and Z are the rotation matrices around the indexed angles respectively.

3.1 Cross-Section Motion

To describe the motion of an arbitrary point on the cross section the previous definition needs to be extended by the effects of transverse shear and warping. Thus the vector r from the origin of the beam relative to a point on the cross-section is constructed by:

- a) the constant axial position x and the elongation $u = u_e + U$, where u_e is due to elastic elongation and U is the elongation due to bending deflections
- b) the transverse shear deformation rotating the cross-section with v around the z -axis and with w around the y -axis
- c) the elastic transversal deformation v and w
- d) the rotation $\theta = \phi + \theta_C/I + \Theta$ around the x -axis, where ϕ is the elastic torsion, θ_C and θ_I is the pitch of the structural and inertial principal axes respectively, and Θ is the rotation due to bending deflections
- e) the position of the point on the rotated cross-section represented by the coordinates η and ζ and the warping displacements W

¹the notation $(\cdot)'$ is used for the derivative with respect to x


 Figure 3: shape functions for u_e and ϕ

 Figure 4: shape functions for v and w

So the vector r with respect to the coordinate system B can be written as:

$$r = \begin{pmatrix} x+u \\ v \\ w \end{pmatrix} + C \begin{pmatrix} 0 \\ \eta \\ \zeta \end{pmatrix} + C \begin{pmatrix} W_1 \\ W_2 \\ W_3 \end{pmatrix} \quad (1)$$

with

$$C = (Z_{-v} Y_{\omega}) C^{EB} \quad (2)$$

3.2 Discretization

The variables of elastic deformation u_e, v, w , and ϕ depend on position and time. To discretize these variables they are separated into space-dependent shape functions $h(x)$ and time-dependent amplitudes $q(t)$. Thus

$$\begin{aligned} u_e &= h_u^T(x) q_u(t) & v &= h_v^T(x) q_v(t) \\ w &= h_w^T(x) q_w(t) & \phi &= h_\phi^T(x) q_\phi(t) \end{aligned}$$

Here h and q are vectors of length N_u, N_v, N_w and N_ϕ where N denotes the degree of freedom for each elastic variable. If $N = 0$ for all degrees of freedom, the elastic beam degrades to a rigid body.

To keep the elastic motion separated from the rigid body motion, appropriate shape functions have to be chosen. Therefore the shape functions need to satisfy the boundary conditions $h(0) = 0$ for elongation and torsion as well as $h(0) = h'(0) = 0$ for bending. This beam element employs algebraic polynomial shape functions as used in CAMRAD II; they are depicted in figure 3 for elongation and torsion as well as figure 4 for bending.

3.3 Equations of Motion

The equations of motion are derived from Hamilton's Principle:

$$\delta \int \mathcal{L} dt = \delta \int (T - U + W) dt = 0 \quad (3)$$

where \mathcal{L} is the Lagrangian, T the kinetic energy, U the strain energy, and W the work of external loads. Those terms will be explained in detail in the following sections. The strain energy is given by the product of stress σ with the strain ε integrated over the volume of the beam:

$$\delta U = \int \delta \varepsilon^T \sigma d\Omega \quad (4)$$

Next, the work of the external loads can be expressed by integrating the body forces b , surface forces t_S and discrete forces F :

$$\delta W = \int \delta p^T b d\Omega + \int \delta p^T t_S d\Gamma + \delta p^T F \quad (5)$$

where p is the position vector of a point on the beam relative to the inertial system (cf. fig. 2). Here surface forces t_S will be discretized and can therefore be treated as discrete forces F . The only body force for the beam model is gravity, which will be treated as inertial force using d'Alembert's principle.

The kinetic energy can be obtained from the integral over density ρ and the absolute beam velocity \dot{p} :

$$\delta T = \delta \int \frac{1}{2} \rho \dot{p}^2 d\Omega \quad (6)$$

Using partial integration in time with $\delta p = 0$ at t_{initial} and t_{final} without loss of generality this can be expressed as (respecting gravitational forces g as inertial force):

$$\delta T = \int \delta p^T (-\ddot{p} + g) \rho d\Omega \quad (7)$$

3.4 Kinetic Energy

Equation (7) can be split into two integrals using the mass per length m :

$$-\delta T = \iint \delta p^T (\dot{p} - g) \rho \, dm \, dx \quad (8)$$

with $p = s + C^{IB}r$, where s is the vector from the inertial system to the beam origin and r is the vector described in equation (1). The corresponding virtual displacement δp can further be written as (with $\delta \psi$ as the virtual rotation of the beam origin)²:

$$\begin{aligned} \delta p &= \delta s - C^{IB} \tilde{r} \delta \psi^{BI} + C^{IB} \delta r \\ &= C^{IB} \begin{bmatrix} I & -\tilde{r} & R_u^T & R_v^T & R_w^T & R_\phi^T \end{bmatrix} \\ &\quad (\delta s^* \quad \delta \psi^{BI} \quad \delta q_u \quad \delta q_v \quad \delta q_w \quad \delta q_\phi)^T \\ &= C^{IB} R^T \delta q \end{aligned} \quad (9)$$

Here R_i represents the Jacobian of the placement r with respect to the degrees of freedom u, v, w , and ϕ . Inserting (9) in (8) the kinetic energy becomes

$$-\delta T = \delta q^T \iint RC^{IB} (\dot{p} - g) \, dm \, dx = \delta q^T M \quad (10)$$

where M is the resulting mass matrix. By differentiating p twice the vector \dot{p} becomes

$$\dot{p} = C^{IB} (\dot{s} + \tilde{\omega}r + 2\tilde{\omega}\dot{r} + \ddot{r} + \omega\dot{s} + \tilde{\omega}\tilde{\omega}r) \quad (11)$$

Neglecting warping and transverse shear effects on inertia equation (1) can be written as:

$$r = x^{EB} + C^{EB} \begin{pmatrix} 0 \\ \eta \\ \zeta \end{pmatrix} = x^{EB} + (Y_{-\beta} Z_\zeta)^T \begin{pmatrix} 0 \\ \eta_b \\ \zeta_b \end{pmatrix} \quad (12)$$

Here η_b and ζ_b identify the cross-section point, relative to the section principle axes at θ_I that are bent but not twisted. Thus the motion of a point on the cross-section is evaluated by

$$r = \begin{pmatrix} x+u \\ v \\ w \end{pmatrix} + \begin{pmatrix} -S_\zeta \\ C_\zeta \\ 0 \end{pmatrix} \eta_b + \begin{pmatrix} -S_\beta C_\zeta \\ -S_\beta S_\zeta \\ C_\beta \end{pmatrix} \zeta_b \quad (13)$$

Consistent with the second order approximation and

$$\begin{pmatrix} \dot{\eta}_b \\ \dot{\zeta}_b \end{pmatrix} = \dot{\theta} \begin{pmatrix} -\zeta_b \\ \eta_b \end{pmatrix} \approx \dot{\phi} \begin{pmatrix} -\zeta_b \\ \eta_b \end{pmatrix} \quad (14)$$

this can be reduced to (15) and derived twice:

$$r = \begin{pmatrix} x+u \\ v \\ w \end{pmatrix} + \begin{pmatrix} -v' \\ 1 \\ 0 \end{pmatrix} \eta_b + \begin{pmatrix} -w' \\ 0 \\ 1 \end{pmatrix} \zeta_b \quad (15)$$

Now equation (15) can be derived twice and thus inserted in equation (11).

²In this paper the notation ($\tilde{\cdot}$) will be used to denote the cross-product matrix

3.5 Strain Energy

The strain energy δU (cf. eq. (4)) is derived from the Green-Lagrange strain tensor, which is obtained by the basis vectors of the undistorted and distorted beam and can be written as

$$f_{mn} = \frac{1}{2}(G_{mn} - g_{mn}) \quad (16)$$

where $g_{mn} = \mathbf{g}_m \mathbf{g}_n$ and $G_{mn} = \mathbf{G}_m \mathbf{G}_n$ are the metric tensors in terms of the curvilinear coordinates $y_m = (x, \eta, \zeta)$ of the undistorted and distorted beam, respectively. The basis vectors are defined as $\mathbf{g}_m = \partial r_i / \partial y_m$ and $\mathbf{G}_m = \partial r_f / \partial y_m$ with

$$r_i = \begin{pmatrix} x \\ 0 \\ 0 \end{pmatrix} + X_{-\theta_c} \begin{pmatrix} 0 \\ \eta \\ \zeta \end{pmatrix} \quad (17)$$

and $r_f = r$ as defined in equation (1). The Green-Lagrange tensor f_{mn} in curvilinear coordinates needs to be transformed into a stress tensor γ_{kl} in local rectangular coordinates in order to apply the constitutive law. Thus the local Cartesian basis z_k with the unit vectors $\mathbf{e}_k = (\mathbf{e}_1, \mathbf{g}_2, \mathbf{g}_3)$ is introduced. The transformation is then given by [10]:

$$f_{mn} = \gamma_{kl} \frac{\partial z_k}{\partial y_m} \frac{\partial z_l}{\partial y_n} \quad (18)$$

with

$$\frac{\partial z_k}{\partial y_m} = \mathbf{e}_k \mathbf{g}_m = \begin{bmatrix} 1 & 0 & 0 \\ -\theta'_c \zeta & 1 & 0 \\ \theta'_c \eta & 0 & 1 \end{bmatrix} \quad (19)$$

The transformation results in the relations $\gamma_{11} = f_{11} + 2\theta'_c(\zeta f_{12} - \eta f_{13})$, $\gamma_{12} = f_{12}$ and $\gamma_{13} = f_{13}$. With the assumption of small strain, $\gamma_{mn} = \varepsilon_{mn}$, where ε is linear in the strain measures. Then, after neglecting all bending and warping terms of higher order as well as warping in y - and z -direction the required strain ε is:

$$\begin{aligned} \varepsilon_{11} &= \frac{1}{2}(G_{11} - g_{11}) + 2\theta'_c(\zeta \varepsilon_{12} - \eta \varepsilon_{13}) \\ &\approx u'_e - \kappa_z \eta + \kappa_y \zeta + 1/2 \phi'^2 (\eta^2 + \zeta^2) \\ &\quad + 2\theta'_c(\zeta \bar{\varepsilon}_{12} - \eta \bar{\varepsilon}_{13}) + \theta'_c \phi' (\zeta \lambda_\eta - \eta \lambda_\zeta) \end{aligned} \quad (20)$$

$$2\varepsilon_{12} = G_{12} - g_{12} \approx 2\bar{\varepsilon}_{12} + (\lambda_\eta - \zeta) \phi' \quad (21)$$

$$2\varepsilon_{13} = G_{13} - g_{13} \approx 2\bar{\varepsilon}_{13} + (\lambda_\zeta + \eta) \phi' \quad (22)$$

In this expression the warping function $W_1 = \lambda \phi'$ has been used.

In order to relate the strain ε to the stress σ used in equation (4) terms for section loads are needed.

Assuming small strain, the sections loads can be expressed as linear combinations of the force strain measure γ and the moment strain measure κ (see ref. [6]):

$$\gamma = C^T \begin{pmatrix} 1 + u' \\ v' \\ w' \end{pmatrix} - \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} \bar{\varepsilon}_{11} \\ 2\bar{\varepsilon}_{12} \\ 2\bar{\varepsilon}_{13} \end{pmatrix} \quad (23)$$

$$\kappa = K - k \quad (24)$$

with $\tilde{K} = C^T C'$, $\tilde{k} = X_{\theta_C} X'_{-\theta_C}$, and $k = (\theta'_C \ 0 \ 0)^T$. It can be shown that $K_x = \theta'_C + \phi'$, so $\kappa_x = \phi$ and $\gamma_x = \bar{\varepsilon}_{11} = u'_e$. Thus the second order approximation for γ and κ is:

$$\gamma = X_{\theta} (u'_e \ -v \ -\omega)^T \quad (25)$$

$$\kappa = X_{\theta} (\phi' \ -w'' - \omega' \ v'' + v')^T \quad (26)$$

For brevity the detailed derivation of the strain measures is omitted here, the reader is advised to refer to references [9] and [7]. The strain equations can now be inserted into the strain energy terms from Hamilton's principle. The stress is determined from strain by the constitutive law $\sigma_{ij} = E_{ijkl} \varepsilon_{kl}$, while only stresses acting perpendicular to the cross-section are taken into account. Thus only σ_{11} , σ_{12} and σ_{13} remain in the energy equations. Now equation (4) can be written in terms of section loads (forces F_i and moments M_i):

$$\delta U = \iint \delta \varepsilon^T \sigma \, dA \, dx \quad (27)$$

$$= \int_0^l [F_x \delta u'_e + F_y 2\delta \bar{\varepsilon}_{12} + F_z 2\delta \bar{\varepsilon}_{13} \quad (28)$$

$$+ M_x \delta \phi' + M_y \delta \kappa_y + M_z \delta \kappa_z] \, dx \quad (29)$$

By integration of $\int dA \delta \varepsilon E \varepsilon$ the section loads can be obtained from stress and hence related to the strain measures. In a next step the transverse shear forces are eliminated from the equations, the shear strain however will still be considered in the material parameters. That leads to the matrix of cross-sectional elastic constants S :

$$\begin{pmatrix} F_x \\ M_x \\ M_y \\ M_z \end{pmatrix} = \begin{bmatrix} S_{uu} & S_{u\phi} + \frac{1}{2}\phi' S_{uu} k_P^2 & S_{uw} & S_{uv} \\ S_{\phi u} + \phi' S_{uu} k_P^2 & S_{\phi\phi} & S_{\phi w} & S_{\phi v} \\ S_{wu} & S_{w\phi} & S_{ww} & S_{wv} \\ S_{vu} & S_{v\phi} & S_{vw} & S_{vv} \end{bmatrix} \begin{pmatrix} u'_e \\ \phi' \\ \kappa_y \\ \kappa_z \end{pmatrix} \quad (30)$$

These factors are required input data for the anisotropic beam model and can be obtained from the previously named beam analysis softwares in section 2. Here k_P^2 is the nonlinear coupling factor, which is the square of polar radius of gyration about the beam axis. For the isotropic model the matrix S reduces to:

$$\begin{bmatrix} EA & \theta'_C E A k_T^2 + \frac{1}{2} \phi' E A k_P^2 \\ \theta'_C E A k_T^2 + \phi' E A k_P^2 & GJ \\ EA z_C & 0 \\ -EA y_C & 0 \\ EA z_C & -EA y_C \\ 0 & 0 \\ EI_{zz} + EA z_C^2 & -EA y_C z_C \\ -EA y_C z_C & EI_{yy} + EA y_C^2 \end{bmatrix} \quad (31)$$

Where y_C and z_C is the horizontal and vertical offset of the tension center and k_T^2 the extension torsion coupling factor.

4 Examples

Calculations of the developed anisotropic flexible beam library have been compared to experimental measurements of the Princeton beam test as well as CAMRAD II simulation results. The static and dynamic behavior of the beam element has been evaluated for deflections, rotations and eigenfrequencies of the beam.

4.1 Static Deflection - Princeton Beam Test

The Princeton beam test [2] is an experimental study of the large static deformation of a cantilevered beam under gravity tip load. It involved an 20x0.5x0.125 inch aluminum beam with a rectangular cross-section. The beam root is rotated around its principle axes so that the tip load is oriented at various angles. Static bending deflections of the tip have been measured as a function of tip load. The softer bending direction is called flap, the stiffer direction chord. The beam is fixed at the root in a way that at zero degrees rotation angle gravitational force deflects the beam chordwise. To compare the experimental results with the implemented beam model a different number of beam segments has been used. The cross-sectional data has been taken from reference [3]. Figure 5 shows the resulting bending deflection in parts of the beam length in flap direction. Figure 6 shows the corresponding results in chord direction.

It can be seen that significant nonlinear effects occur with increasing tip loads. Using one or two beam

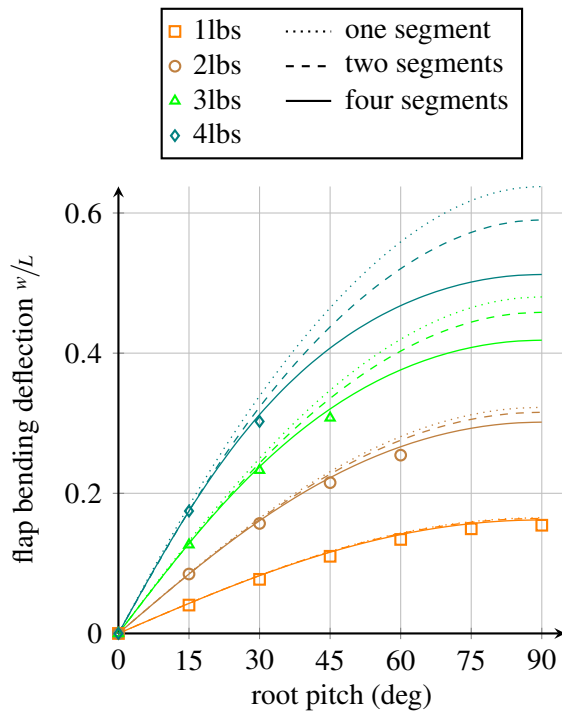


Figure 5: Princeton beam test: Comparison of flap-wise tip bending deflection of measured and calculated (marks = experimental data)

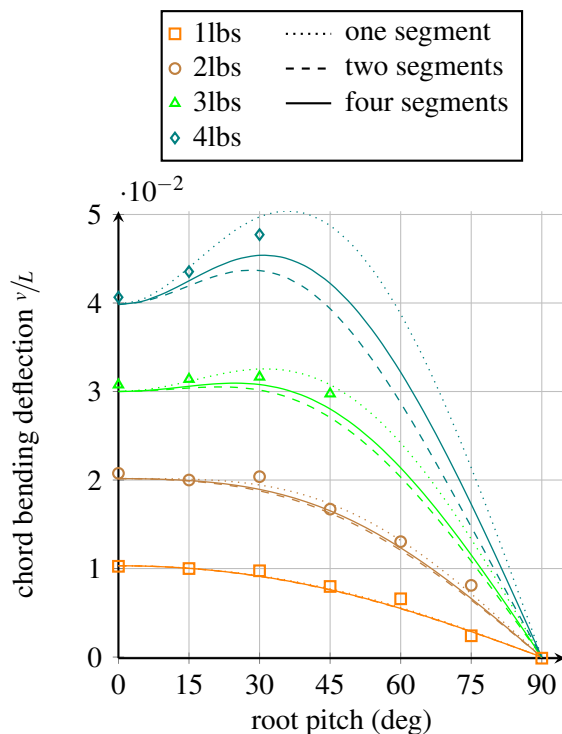


Figure 6: Princeton beam test: Comparison of chord-wise tip bending deflection of measured and calculated (marks = experimental data)

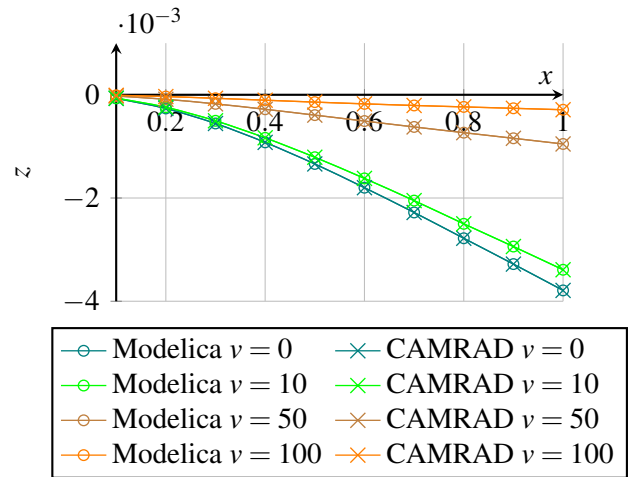


Figure 7: Vertical deflection of the Princeton beam at different tip speeds $v[m s^{-1}]$

segments the simulation is not capable of reproducing the experimental results due to the second order approximation. Using four beam segments however the large nonlinear deformation is captured by the rigid body motion, which is exact. In that case the simulation gives good results for flap and chord deflection.

4.2 Dynamic Behavior

To validate the dynamic behavior of the library a cantilever beam segment is rotated around its vertical axis at root using different tip speeds $v = \Omega r$, with Ω as the rotational speed. The transversal deflections in x, y, z -direction as well as the rotation angles $\theta, -\beta, \zeta$ around the x, y, z axis respectively are measured with ten virtual sensors at equally spaced positions along the principal axis. To test the isotropic behavior cross-sectional data from the Princeton beam test have been used. In order to analyze the anisotropic characteristics of the beam these parameters are expanded with structural and inertial coupling factors. This way more than 35 test beams have been created to vary all possible input parameters and compared the virtual Modelica measurements to CAMRAD II simulation results. Exemplary figure 7 depicts the vertical deflection of the isotropic Princeton beam. It can be shown, that the deviations between the two simulation softwares for all isotropic and anisotropic test cases are smaller than the predefined numerical tolerance of $e = 10^{-5}$.

In rotorcraft analysis a widely used tool is a fan plot. They show the relation between the rotary speed and the eigenfrequency of a rotor. Again results are compared to CAMRAD II. The eigenfrequencies of the Modelica simulation are obtained using the "lin-

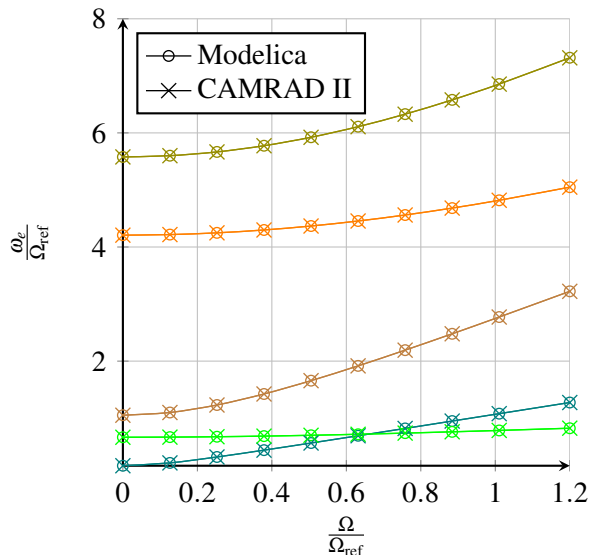


Figure 8: Fan plot of the first five eigenfrequencies for the Princeton beam. Abscissa: normalized eigenfrequency ω_e ; Ordinate: normalized rotary speed Ω ; reference speed $\Omega_{\text{ref}} = 380 \text{ rad s}^{-1}$

earizeModel" function and extracting the eigenvalues with the Linear Systems Toolbox. Figure 8 shows the variation of the first five eigenfrequencies at different rotary speeds normalized with the reference speed of $\Omega_{\text{ref}} = 380 \text{ rad s}^{-1}$. For all frequencies it can be shown that the results of both simulation software match perfectly.

4.3 Animation

The implemented anisotropic flexible beam library is capable of visualizing the deformations of the beam. For this purpose the Modelica surface visualizer from the MultiBody library is employed. Thus all standard features such as colors, transparency etc. are available. To make small deformations visible an amplification factor has been implemented which exaggerates all deformations and rotations of the beam segment in the animation window. To save computational power the resolution of the animation can be reduced or completely disabled. An example of the animation is presented in figure 9. Here four flexible beam elements are used to simulate a helicopter rotor blade (shown with exaggerated amplitudes), yet the consideration of aerodynamic forces is ongoing work.

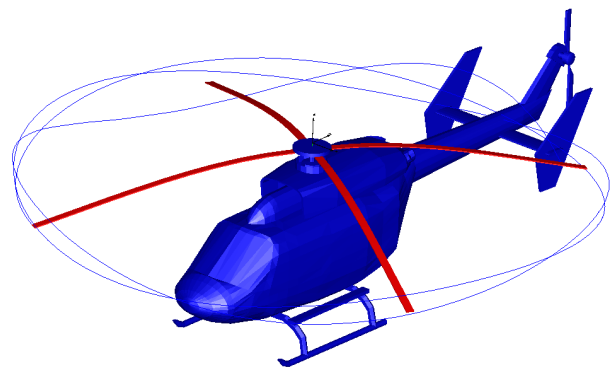


Figure 9: Animation of four elastic beam segments used as helicopter rotor blades

5 Conclusions

This paper presents a structural dynamic library to model anisotropic and nonhomogeneous elastic beams in Modelica. It is capable of simulating nonlinear extensional, torsional and flexural deformation and the couplings between those degrees of freedom. Using cross-sectional modeling theory the user is able to provide different varying material parameters along the beam principal axis. The results correlate with experimental beam measurements as well as other beam simulations software. To model large nonlinear deformation multiple beam segments can be connected in series.

References

- [1] CESNIK, C. E. S. and D. H. HODGES: *Variational-Asymptotical Analysis of Initially Curved and Twisted Composite Beams*. Applied Mechanics Review, 46(11), 1993.
- [2] DOWELL, E. H., J. TRAYBAR and D. H. HODGES: *An experimental study of the nonlinear stiffness of a rotor blade undergoing flap, lag and twist deformations*. NASA, 1975.
- [3] DOWELL, E. H., J. TRAYBAR and D. H. HODGES: *An experimental-theoretical correlation study of non-linear bending and torsion deformations of a cantilever beam*. Journal of Sound and Vibration, 50(4):533–544, 1977.
- [4] FRIEDMANN, P.P, B. GLAZ and R. PALACIOS: *A moderate deflection composite helicopter rotor blade model with an improved cross-sectional*

- analysis*. International Journal of Solids and Structures, 46(10):2186–2200, 2008.
- [5] GIAVOTTO, V. and M. BORRI: *Anisotropic Beam Theory and Applications*. Computers and Structures, 16(1-4), 1983.
- [6] HODGES, D. H.: *Nonlinear composite beam theory*. American Institute of Aeronautics and Astronautics, 2006.
- [7] JOHNSON, W.: *CAMRAD II, Comprehensive Analytical Model of Rotorcraft Aerodynamics and Dynamics*, 1992-1997.
- [8] JOHNSON, W.: *Rotorcraft dynamics models for a comprehensive analysis*. 1998.
- [9] SPIESS, C.: *Entwicklung eines anisotropen Strukturmodell zur Simulation von elastischen Rotorblättern*. Diploma Thesis, Technische Universität München, Garching, 2011.
- [10] WASHIZU, K.: *Variational methods in elasticity and plasticity*. Pergamon press Oxford, UK, 1975.

Modeling and Simulation of a Fault-Tolerant Electromechanical Actuation System for Helicopter Swashplates in Modelica

Sebastian Seemann
EADS Innovation Works
TCC6 Energy and Propulsion
81663 Munich, Germany
sebastian.seemann@eads.net

Clemens Schlegel
Schlegel Simulation GmbH
Meichelbeckstr. 8b
85356 Freising, Germany
cs@schlegel-simulation.de

Abstract

Replacing hydraulic primary flight control actuators by electromechanical actuators imposes the problem of reduced reliability. This problem may be overcome by using redundant actuators what in turn increases the system complexity. The appropriate redundancy level and component mapping must be assessed. In specific failure cases the system must be reconfigured in order to maintain the specified performance level to meet aircraft safety regulations. The assessment of the system's reaction upon such kind of scenarios is however a complicated task and must be supported by modeling and simulation. Therefore, modeling and simulation of such a fault-tolerant electromechanical system in Modelica is described in this paper. Sample simulation results are presented and discussed.

Keywords: electromechanical actuator; redundancy; fault-tolerance; over-determined kinematics; helicopter; swashplate; flight controls;

1 Introduction

A general trend in aviation is to replace hydraulic subsystems like primary flight control actuators by electromechanical devices. However, substituting a hydraulic actuator by an electromechanical actuator (EMA) has the disadvantage of reduced component reliability. This accompanies two major challenges. First, in order to meet aircraft safety regulations higher degrees of redundancy are needed for the utilization of EMAs. Moreover, in the case a redundant actuator jams mechanically, it must be disconnected from the swashplate to maintain controllability of the

remaining actuators and the ability to position the entire swashplate.

The system under investigation is therefore specified to provide fail-operative behavior for major mechanical failures and dual-fail-operative behavior for combinations of any other failures. This requires certain degrees of redundancy of all system parts and meaningful mapping of the components in order to allow for failures while maintaining function and performance. Furthermore, suitable means for failure detection, failure isolation and system reconfiguration are needed.

2 System architecture and component failures

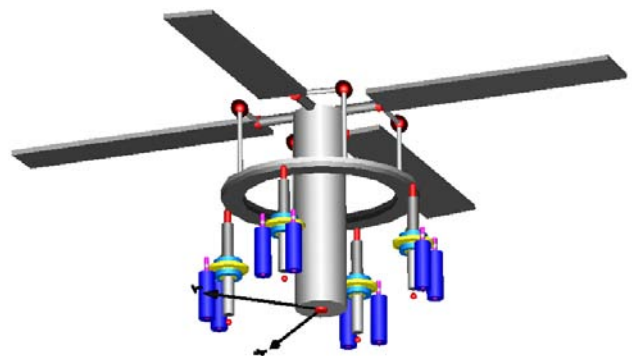


Figure 1: Swashplate actuation system

The concept investigated comprises four vertically arranged and equidistantly spaced actuators for the operation of a three degree of freedom helicopter swashplate, each of them containing two motors (see figure 1, blue cylinders). The system operates against aerodynamic forces caused at the rotor blades and exerted on the rotating upper ring of the swashplate through pitch links. The stationary lower ring of the

swashplate is positioned by the four EMAs. All actuators are simultaneously active to achieve a minimum of nominal loading. The provided redundancy allows for the malfunction of one actuator unit, the three remaining EMAs safely continuing control of the swashplate with reduced performance.

2.1 Swashplate actuator

Each swashplate actuator consists of two electric motors in torque-summing configuration and a mechanical drive train. The latter comprises a two-stage gearbox, ballscrew and nut assembly, and an output piston to the swashplate attachment. The variety of conceivable mechanical failure modes can be categorized into two types of mechanical failures to be taken into account, namely fracture and jamming of the drive train.

For monitoring and control purposes each single actuator drive path is equipped with an absolute position sensor and two cut force sensors. Moreover, each of the two electric motors per actuator features sensors for angular position, phase currents, and temperature.

2.2 Disconnect device

Under all flight conditions, the swashplate must be controllable in three degrees of freedom, i.e. collective, pitch, and roll (see e.g. [1]). As mentioned, the risk of a mechanical jam must be considered which can be caused, for instance, by wear or debris. To avoid the swashplate getting stuck due to a single jammed actuator, fail-safe degradation of the overall actuation system is needed. For this reason, each actuator is fitted with a disconnect device, decoupling the output shaft from the mechanical drive train [2]. After disconnection of one actuator the swashplate is still safely controlled by the remaining three actuators. However, the time needed for disconnection is critical regarding stability and stress and therefore imposes strict requirements on failure detection and disconnect activation.

2.3 Electric motor

The most common design for electrically driven flight surface actuators is a permanent magnet synchronous motor (PMSM) fed by pulse-width modulated (PWM) inverters. This is due to the superior torque and power density of such devices.

The most common faults are device failure within the inverter and open and/ or short circuit failures in the motor windings. This failures typically lead to a loss of motor output torque (open circuit failure or

inverter failure) or a drag torque induced by short-circuit currents.

2.4 Power supply

The electric power sources driving the motors are also critical components of the overall system. The required power supply reliability is ensured by a multi power bus configuration. The system has four independent power supplies, each being connected to one actuator control electronics (ACE) unit. The failure cases considered include a loss of power supply output power, and out-of range output voltage.

2.5 Redundancy and component mapping

The maximum accepted probability of catastrophic events of an aircraft system is $1 \times 10^{-9} \text{h}^{-1}$ [3]. To meet this figure several subsystems must be redundant and the connections of subsystems must be designed such that a single failure results in minimum system degradation.

Regarding the overall drive train a static redundancy approach is followed, i.e. all actuators are simultaneously active. Each is driven by two fully independent paths of torque generation, comprising electric motors, power supply busses, power electronics, and control computers. In order to minimize system degradation after a failure the two motors of a single actuator are controlled by different ACE units. Moreover, each actuator has a different combination of motor control electronics assignment in order to avoid the loss of two entire actuators after two ACE failures.

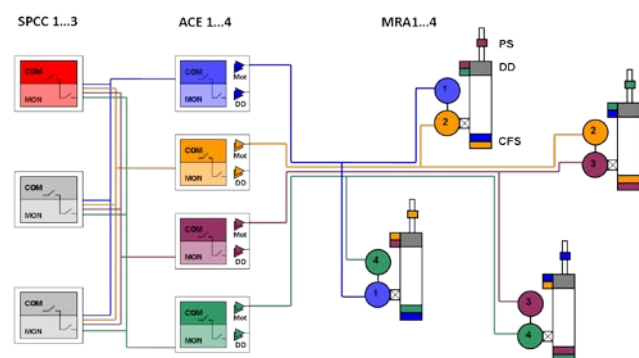


Figure 2: System architecture

Figure 2 shows the applied component mapping. The boxes on the left-hand side represent the topology of the dual-lane computers, namely swashplate control computer (SPCC) and actuator control electronics. Since motor control responsibilities are split and mapped to all four ACEs, they are operating in an active/active configuration. The SPCC functions can be assumed to be functionally integrated in a flight

control computer (FCC) in a master/slave configuration and therefore its topology will be adopted. Three SPCCs are depicted being the minimum viable degree of redundancy. An ACE additionally contains power stages for motor operation and disconnect device activation, respectively (see triangles in figure 2). In addition to the two motors (circles), each main rotor actuator (MRA) comprises a disconnect device (DD) equipped with dual activation path, two cut force sensors (CFS) and a single position sensor (PS). As can be seen from the respective color coding, the disconnect device is controlled by two ACEs different to those assigned to the two motors of an actuator. This is to allow for disconnection even after both motors were lost due to ACE malfunctions to decouple dead rotary inertia.

3 Control and monitoring

In this paragraph the control and monitoring approach is briefly introduced. A more detailed description can be found in [4].

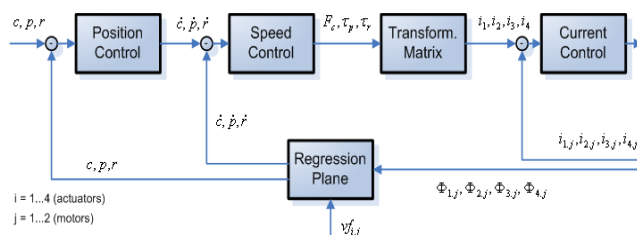


Figure 3: Control architecture

3.1 Control architecture

The presented actuator arrangement causes over-determined kinematics, since four actuators are used to control the three degrees of freedom of the helicopter swashplate (collective, pitch, and roll). A control approach is used which is based on transforming the four actuator position signals into three position parameters, derived from a method introduced by [5]. Control is performed by means of a cascaded PID architecture comprising current, speed and position loop for each of the three directions (figure 3). Eight motor position signals provide position feedback (two resolvers per actuator; the absolute position sensor on actuator level is used for monitoring only). By means of a regression plane the actual swashplate position is determined and transformed into respective actual collective, pitch, and roll values. The three force/torque set values are transformed back into four actuator torque set values, i.e. one per actuator. Hence, each two motors per actuator receive a common torque command. By this set-

up force fighting between single actuators is excluded by design for nominal conditions. This approach based on coordinate transformations is a simple and powerful method, which is however threatened if specific failures are not detected.

3.2 Monitoring architecture

In order to mitigate the effects of the component failures described above, the system must be fitted with appropriate monitoring. As a general philosophy, simple mechanisms are desired. Therefore, most of the monitoring algorithms rely on redundancy of information and signal comparison. Complex health-monitoring and the associated knowledge database are avoided. In addition, the control functionality is totally decoupled from fault-detection algorithms for its continuous operation. In other words, control loops are never influenced by ongoing fault detection processes, unless an unambiguous decision was made by the monitoring part.

There are three reconfigurations designed to be automatically executed by the system, namely isolation of faulty position signals, disconnect device activation, and motor shutdown. For this purpose, the monitoring subsystem supervises the sensor signals of all actuators, compares redundant information and generates trusted signals fed to the controllers. Five monitors are continuously assessing parallel tasks:

- Sensor monitor
- Actuator positioning monitor
- Swashplate positioning monitor
- Jam monitor
- Motor monitor

An additional decision layer evaluates the opinions of the independent monitors and initiates the respective reconfiguration. In case a faulty position signal is detected, the respective signal is permanently isolated by excluding it from the regression plane computation by means of a validity factor (see also [4]). The disconnect device is activated via the respective power stages (see figure 2) if a mechanical jam was unambiguously detected. Motor failures are typically detected internally by their dedicated control electronics.

4 Model implementation

4.1 General modeling approach

For model implementation the Modelica [6] based simulation software Dymola [7] is used. The overall

system simulation model is shown in figure 4. Within the blue dotted frame the system components are located (top down order): The lower swashplate (including the inertias of the upper swashplate and the rotor blades) and the respective actuator hinges at the helicopter structure, the array of actuators, and the control and monitoring blocks. Control (green) and monitoring (orange) loops are depicted. Inputs to the system are aerodynamic forces, power supply and position commands. On the bottom of figure 4 the system parameters are illustrated, assigned to the five categories mechanical drive train (MDT), power stages and motors (PSM), position commands (POS), external forces (FORCE), and failure injection (FAIL).

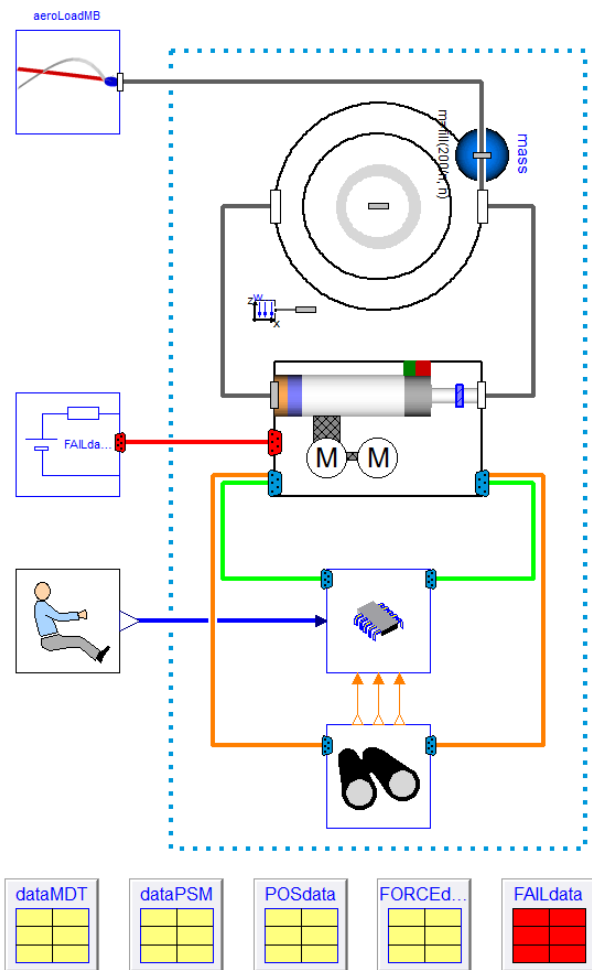


Figure 4: Top layer of system simulation model

A major idea of the simulation model is to investigate not only one specific system design, but to allow for comparison of the performance of several concepts against each other. One important goal of the model therefore is easy generation of models of concept variants. Therefore, for instance the number

of actuators is a model parameter in order to allow for variation of the actuator redundancy. The actuators are grouped in an array of components with the respective connectors. Figure 5 illustrates concepts comprising three, four, and five actuators, respectively.



Figure 5: Swashplate actuation design variants

Apart from the Modelica Standard Library no other publically available model library has been used. Class parameterization is applied for the handling of different models of the same component (e.g. drive train with and without friction) and of predefined sets of parameters, e.g. aerodynamic loads, command inputs, and failure cases. Via inheritance fully parameterized simulation experiments were stored, thus facilitating the handling of the large number of simulation test cases to be assessed. The Modelica feature of arrays of components proved to be an essential advantage for the implementation of redundant components. In the following paragraphs the global model components are described.

4.2 Electromechanical actuator

As mentioned, an electromechanical actuator consists of a mechanical drive train (including disconnect device), two motors and a power inverter. The disconnect device implementation is based on constraint forces rather than friction: In connected state, internal forces are computed which inhibit relative movement of the disconnect device input and output connections. After activation no more force is transmitted, both parts move independently. The EMA failure cases, namely drive train jamming and drive train fracture, are modeled by activation of a brake and deactivation of a clutch.

For assessment of the effects of mechanical losses in the drive train two implementations have been realized: Friction forces and torques may degrade the overall system dynamics and must therefore be contained in the simulation model. For investigation of the effects of mechanical losses on power consumption an efficiency model has been implemented as an alternative, avoiding the numerical issues and computational load of friction models.

4.3 Motor and inverter

The inverter and motor models are built according to the functional modeling layer specifications [9]. This allows for improvement of the overall system model computational efficiency by exclusion of high frequency switching behavior and reduction of the motor and associated controls model based on the principles described in [8]. Motor controls are implemented using standard space vector control structures with a decoupled control of the current flux- and torque components.

4.4 Monitoring and control

The monitoring concept and the control algorithm have been described above. Both are implemented in single model components connected to the array of actuator components. In contrast to the real system implementation the simulation model does not contain redundant computers. The effect of an ACE computer failure can be emulated by switching off the respective power supply. Swashplate control computer (SPCC) topology and failure detection are outside the scope of this paper. The monitoring algorithms are implemented as a sampled block as it would be implemented in flight hardware. Even though an analogue implementation would be preferable for simulation performance reasons, a sampled implementation is required for future hardware-in-the-loop simulations.

4.5 Aerodynamic forces

The aerodynamic forces acting on the swashplate and its actuators are given as a sequence of signals assigned to a matrix of flight conditions, e.g. stabilizing, high rate pull up, 30° turn with severe turbulences, etc.

4.6 Power supply mapping

The power supplies are mapped to motors according to the assignment illustrated in figure 2: Each power unit supplies two motors containing to different actuators. Thus the failure of a single power unit does not cause loss of a whole actuator. A dedicated mapping algorithm allows automated mapping of the parameterized power supplies and actuators. It is implemented as variable loops of connect statements.

4.7 Failure injection

The simulation model covers a set of relevant failure cases, as they were already introduced above. Table

1 shows a summary of the most relevant component failures and indicated the manner of injection. Each injection is parameterized via setting of a pair of time/ value. Since not all combinations of failures and fail sequences are relevant, predefined sets of parameterizations have been defined as parameter records. All of them are collected in a failure record on the top model hierarchy, while class parameterization allows activation of specific fail cases.

Component	Failure	Injection
Mechanical drive train	Fracture	Clutch
	Jam	Brake
Motor-Converter	No torque	Switch
	Short circuit	Switch
Power bus	Power interruption	Switch
	Voltage drop	Switch
ACE	Shutdown	Power switch
Force sensor	Freeze	Signal Hold
	Bias	Signal Add
Position sensor	Freeze	Signal Hold
	Bias	Signal Add

Table 1: Component failures covered in the current model

5 Simulation results

Validation of the system behavior requires a large amount of simulation test cases to be performed. Those are implemented by means of a dedicated test case library which can be re-run on demand. This allows for comparability, consistency and easy reproduction of the total set of test cases. This chapter presents a short selection of simulation results of the most relevant failure cases.

5.1 Motor failure

Figure 6a shows the response of the system to a collective position demand signal injected at $t=0.5s$. Both motors of actuator 1 need the same current (fig. 6b) and deliver the same torque (fig. 6c). At 0.6s a winding short circuit failure of motor 1 occurs. Motor 2 now draws more current and delivers nearly double torque, whereas faulty motor 1 shows short-circuit current but only a small braking torque, while the actuator speed is maintained. At $t=1s$ the position demand is satisfied, the motors continuously reduce speed. Consequently, the speed dependant short-circuit current of motor 2 almost disappears after

1.5s. Motor 2 keeps the actuator in steady state. For this simulation the motor monitoring was deactivated in order to check the actuator performance in the case of a motor failure.

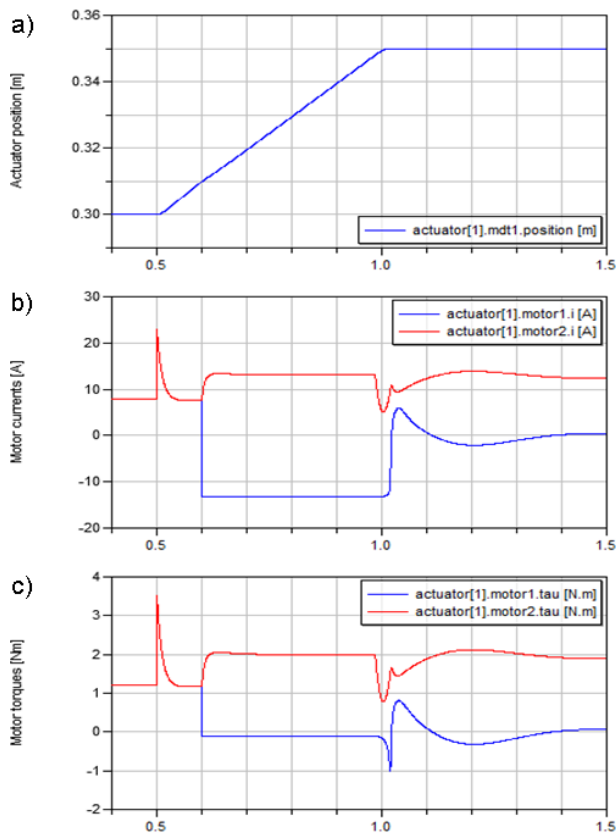


Figure 6: Actuator response on motor short circuit failure

5.2 Power failure

Figure 7a shows the response of the system to a collective position demand signal injected at 0.5s. All power units deliver the nominal voltage of 270V, all motors draw the same current. At $t=0.6s$ a first power supply fails. Motor 2 of actuator 1 and motor 1 of actuator 2 subsequently draw no more current (and deliver no torque). Motor 1 of actuator 1 and motor 2 of actuator 2 compensate for this loss, i.e. draw the double current while the actuator speed is maintained.

At $t=1.1s$ a second power supply fails, thus only 4 out of 8 motors of the overall system remain active. In figure 7c motor 1 of actuator 1 represents the 4 operative motors, the other three of which are not displayed for transparency reasons. The demanded position is maintained, but the system behaves more sensitively. Damping of the current oscillations caused by the second power unit failure requires almost one second. The system remains operational with reduced performance.

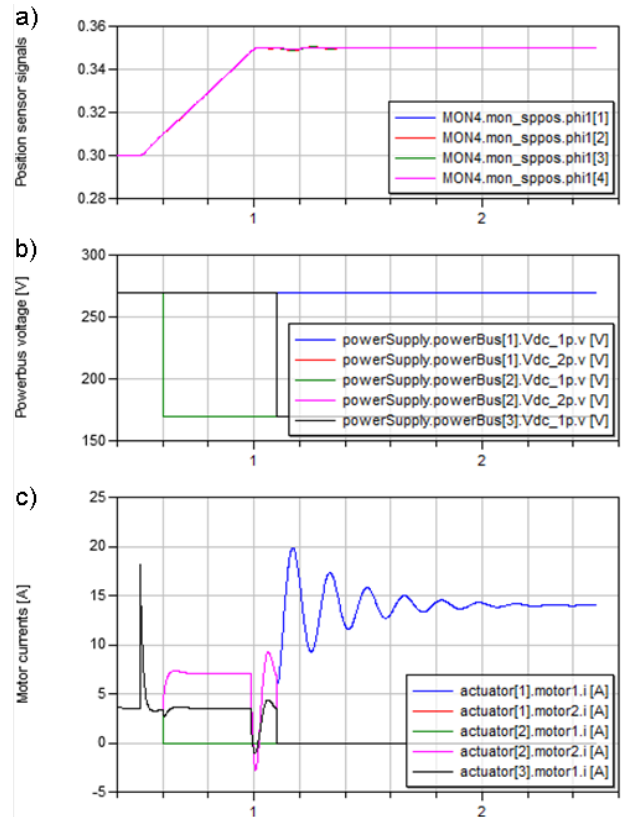


Figure 7: System response on single and double power failure

5.3 Mechanical jam

For the jamming scenario the following critical condition is simulated: A sudden friction force causes instantaneous jamming, i.e. inability of displacement. This exposes the system to the most stringent requirement regarding detection time. The disconnect device must be activated rapidly to maintain control stability and limit mechanical stress.

Figure 8 shows the respective simulation results. The swashplate performs a collective movement of 0.05m starting at $t=0.5s$ with maximum speed $v_{max}=100mm/s$. At $t=0.7s$ a mechanical jam is injected at actuator 1 (fig. 8a, blue lines). Current commands immediately change, expecting mainly actuator 1 to compensate for the position control deviation. This effect highlights the dependency of the controller on sophisticated jam detection: Of all actuators the failed one is powered most, which in turn weakens the remaining operative EMAs. The measured forces however illustrate that actuator 2 and 4 (pink line) sustain almost the full loads, while actuators 1 and 3 do not contribute significantly.

The low measured force at actuator 1 however contradicts to the high commanded current. This effect is used for jam detection by means of an internal torque residual. At $t=0.9s$ the disconnect device, represented by an idealized mechanical clutch, is acti-

vated. The converging position signals show that horizontal washplate attitude is recovered within 0.2s. Subsequently, for geometrical reasons, actuators 2 and 4 are in charge of sustaining the washplate loads, while actuator 3 draws current just for stabilization. The motors of the failed actuator are shut down.

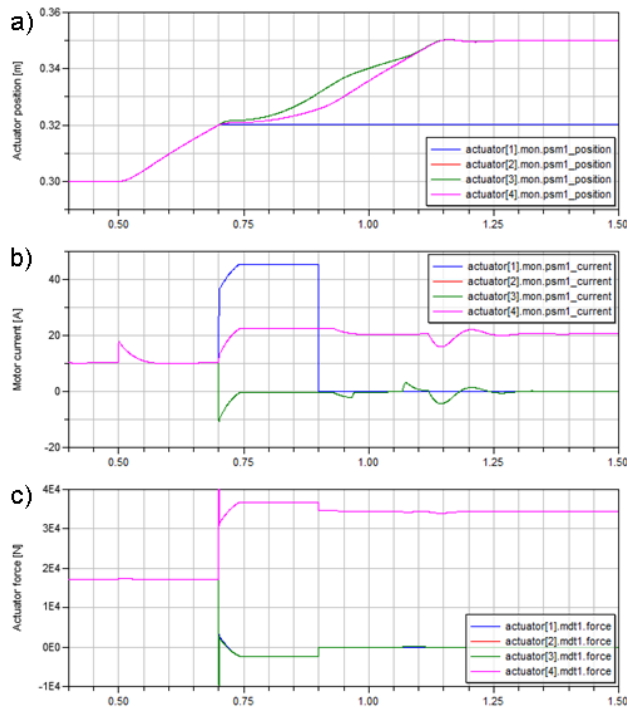


Figure 8: System response on mechanical jam

5.4 Position signal failure

Detection and isolation of a failed position sensor is a very important task, since the measured positions are the only signals directly influencing the control loops. Via the regression plane, a non-detected position sensor failure would lead to faulty feedback to the position and speed loop.

Figure 9 depicts signals related to a position sensor failure and its detection. Again, a collective washplate movement of 0.05m is commanded at $t=0.5s$ with full specified speed. Resolver 1 at actuator 1 fails due to freeze at $t=0.6s$. This measured value is however be taken into account for the feedback calculation of the washplate position.

As can be seen from figure 9a, the false position value leads to an increasing diversion of the washplate from the horizontal plane. In the washplate positioning monitor the distance of each measured position to the overall reference plane is calculated based on the Hesse normal form. Figure 9b shows that the faulty sensor 1 deviates faster than the others. After exceeding a predefined deviation threshold, this is

considered a sensor failure. To avoid that temporary disturbances may lead to a false decision, several confirmation cycles are performed (see figure 9c). At $t=0.8s$ the decision is confirmed and resolver 1.1 is isolated. As an immediate effect, all remaining position signals perfectly fit to the plane calculated without the failed signal (see figure 9b). Horizontal attitude of the washplate is recovered and maintained as shown in figure 9a.

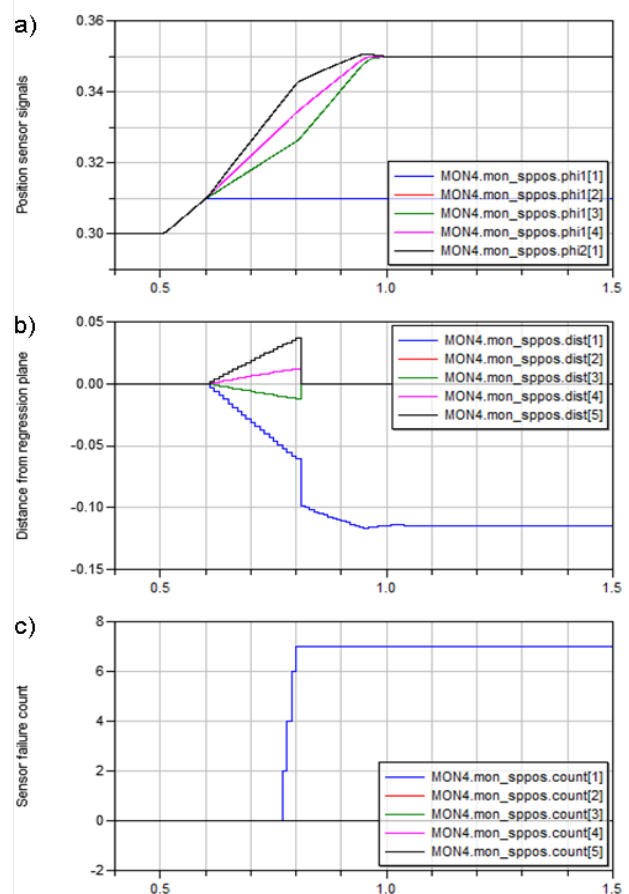


Figure 9: System response on position sensor failure

6 Conclusion

The presented paper introduced a safety-critical application of electromechanical actuators. The accompanied challenges of such system were described and a summary of relevant failure cases was given. Modelica is considered a suitable means for modeling of this kind of system including the specific characteristics, such as redundancy, mapping, fault injection, failure detection, and reconfiguration. Exemplary simulation results depicted the system response on specific relevant failure cases. It was shown that performance and reconfiguration behavior are as expected.

Acknowledgements

The described investigation is funded by the European Union's Seventh Framework Programme (FP7/2007-2013) for the Clean Sky Joint Technology Initiative under grant CSJU-GAM-SGO-2008-001. The contribution of the motor and inverter models by the project partner University of Nottingham is highly appreciated.

References

- [1] W. Bittner. Flugmechanik der Hubschrauber, Springer Verlag, 2005.
- [2] A. Naubert. Pyrotechnic Jam-Relief Mechanism for Electromechanical Actuators in Flight Control Applications, Proceedings of the Actuator12 International Conference, Bremen, Germany, 2012
- [3] European Aviation Safety Agency. Certification Specifications for Large Rotorcraft (CS29), Köln, Germany, 2003
- [4] S. Seemann, M. Christmann, P. Jänker. Control and Monitoring Concept for a Fault-Tolerant Electromechanical Actuation System, Proceedings of the R3ASC International Conference, Toulouse, France, 2012
- [5] B. K. Walker, E. Gai. A New Approach to Fault-Tolerant Helicopter Swashplate Control, AIAA Aircraft Design, Systems and Technology Meeting, Forth Worth, USA, 1983
- [6] www.modelica.org
- [7] www.3ds.com/de/products/catia/portfolio/dymola
- [8] T. Wu, S. Bozhko, G. Asher, P. Wheeler. Fast Reduced Functional Model of Electromechanical Actuators for More-Electric Aircraft Power System Study, SAE Technical Paper 2008-01-2859, Nov.2008
- [9] S. Bozhko, T. Wu, C.I. Hill, G. Asher. Accelerated simulation of complex aircraft electrical power system under normal and faulty operational scenarios, in Proc. IEEE IECON 2010, Nov.2010, pp 333-338

Survey of appropriate matching algorithms for large scale systems of differential algebraic equations

Jens Frenkel¹ Günter Kunze¹ Peter Fritzson²

¹Dresden Technical University, Institute of Mobile Machinery and Processing Machines

²PELAB - Programming Environment Lab, Dept. Computer Science

Linköping University, SE-581 83 Linköping, Sweden

{jens.frenkel, guenter.kunze}@tu-dresden.de,

peter.fritzson@liu.se

Abstract

This paper presents a survey on matching algorithms which are required to translate Modelica Models. Several implementations of matching algorithms are benchmarked on a set of physical models from mechanical systems in ODE and DAE representation. The major part of algorithms is based on the Augmenting Paths Method and one algorithm is based on the Push-Relabel Method. The algorithms are implemented in the programming language C and Meta-Modelica. In addition two cheap matching algorithms are used to jump-start the advanced matching process.

Keywords: matching; index reduction; modelimark

1 Introduction

A major benefit of Equation based Object Oriented modeling Languages (EOOL) like Modelica is the possibility of acausal modeling. It increases the reusability of models and simplifies the description of physical systems. In order to simulate an acausal model, all equations have to be transformed and sorted yielding a causal model description. The process of transforming equations into assignments is thus called *causalization*. The main task of causalization is to match each equation to a variable. It is one of the most important challenges of any EOOL compiler.

Most models from EOOL give rise to very large and sparse differential algebraic equation (DAE) systems [19],[20],[25]. The challenge of the matching process is therefore to transform the model into an ordinary differential equation (ODE), so that it can be solved through the application of standard numerical time integration algorithms.

Pantelides [21] provides an algorithm to get a

so called *perfect matching*, transforming the system to block lower triangular form (BLT) providing all necessary information to apply index reduction and thereby transforming a DAE into an ODE. Driven by the need of numerical stability several index reduction algorithms have been developed in the past [16],[18],[19],[20],[25],[27].

There are other matching algorithms next to those presented by Duff [4]. They can be divided into different classes of worst case time complexities. The most common complexities are shown in Figure 1¹².

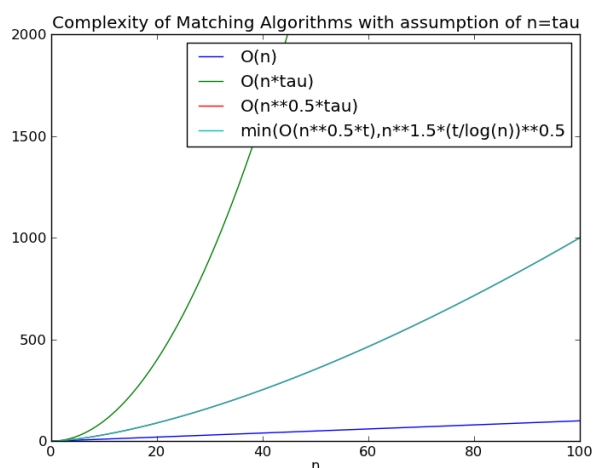


Figure 1: Typical worst case complexities of matching algorithms [9]

Since more powerful computers allow for larger models with more equations, a future challenge will be to optimize the scaling of EOOL compilers with respect to model size. As shown in [11] the effort of state of the art EOOL compilers is proportional to the

¹n: Number of Equations

² τ : non zero entries in the Adjacency Matrix

second or even the third power of the number of equations, depending on the model structure. Thus it is worth studying how the combination of matching and Pantelides Algorithm can be further optimized.

The next section provides a brief introduction to matching theory and index reduction. It is followed by an overview on selected matching algorithms based on augmenting paths and the push relabel technique. Section 4 discusses the possibility to combine the matching algorithms with index reduction by looking at some examples. A comparison of runtimes of all these algorithms is presented in section 5 followed by a discussion and concluding remarks in section 6.

2 Theory of Matching and Index Reduction

2.1 Matching Theory

The aim of this section is to give an introduction to the general definitions of matching algorithms. For further information, the reader is referred to [5],[6],[8],[9],[2],[10]. As mentioned above and shown in detail by Elmquist [10] matching algorithms provide the information how a system of equations can be transformed symbolically into a system of assignments. The mathematical idea behind this, is to transform the system into block lower triangular (BLT) form and to solve it by a simple forward substitution process [5]. As Duff proposed in [5] the transformation to BLT form is split into two stages:

- Match each equation to a variable and transform the problem description into a directed graph
- Find a traversal of the directed graph which means to sort the equations and identify algebraic loops

For the second step Tarjan's Algorithm [26] is very efficient and offers time linear complexity with respect to the number of equations [6]. To understand the first step one has to look at the *Adjacency Matrix* of a system of equations. The rows of the Adjacency Matrix correspond to the equations whereas the columns correspond to the variables of the system. The Adjacency Matrix has an entry (=1) at row i and column j , iff equation i contains variable j . The number of entries in the Adjacency Matrix is denoted with τ . For a nonsingular system, the matching algorithm finds an unsymmetric permutation which produces a zero-free main diagonal. The set of all nonzero entries on the

main diagonal is called a *transversal*. A set containing the maximum number of nonzero elements is called a *maximum transversal*. A simple example is shown in Figure 2.

$$\begin{array}{l}
 a) \quad b+c = 0 \\
 \quad \quad a = 10 \\
 \quad \quad a+c = 2
 \end{array}
 \quad
 b) \quad \begin{pmatrix} 0 & \mathbf{1} & \mathbf{1} \\ \mathbf{1} & 0 & 0 \\ 1 & 0 & \mathbf{1} \end{pmatrix}
 \quad
 c) \quad \begin{pmatrix} \mathbf{1} & 0 & 0 \\ 1 & \mathbf{1} & 0 \\ 0 & 1 & \mathbf{1} \end{pmatrix}$$

Figure 2: Equation System (a) with Adjacency Matrix (b) and permuted matrix in BLT form (c) from matching highlighted in boldface.

The Adjacency Matrix can also be presented as a bipartite graph with one set of nodes representing equations (green) and another representing variables (yellow). The edges of the graph represent the nonzero entries in the Adjacency Matrix. For the simple example presented above in Figure 2 the bipartite graph is shown in Figure 3.

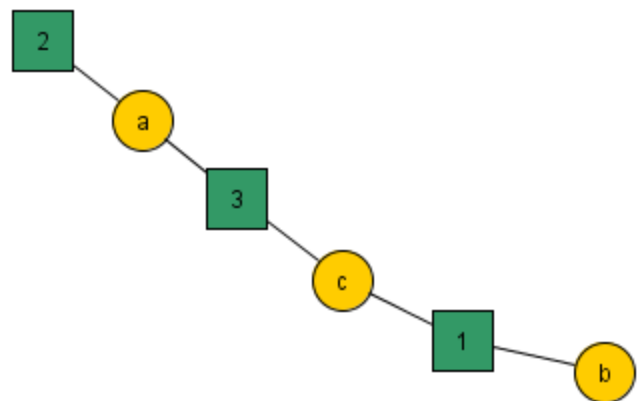


Figure 3: Bipartite graph for the example from Figure 2

A set of matched equations and variables is called *matching* or *assignment block*. If no additional matches can be found, the matching is called *maximum*. In case of a square matrix the matching is *complete (perfect)* if all equations are matched. In case of a non-square matrix the matching is complete if either all equations or all variables could be matched. A sequence of connected nodes is called *path*. If each of the nodes on a path belong to the matching, then it is called an *alternating path relative to an assignment*. If the alternating path has an unmatched equation at one end and an unmatched variable at the other end it is called a *augmenting path*. In such a case the matching could be increased by one if all assignments from the

path are removed from the matching and all other assignments from the path are added. This procedure is called *reassignment* or *rematching* [8].

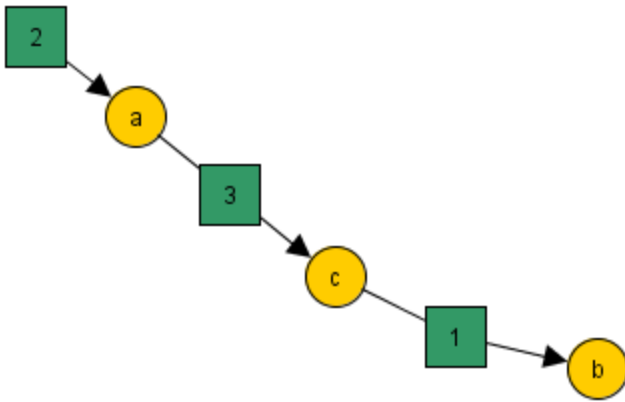


Figure 4: Matched Bipartite graph for the example from Figure 2 with alternating path $M = \{(2,a),(3,c),(1,b)\}$

2.2 Index Reduction

In case of a DAE system with differential index $v_d > 1$ [[27], Definition 2.1] no complete matching can be found. If the system is not structurally singular an appropriate symbolical index reduction algorithm must be employed to reduce the differential index v_d to at least one.

As mentioned in [27] and [25] several symbolical methods for index reduction are available. The graph-theoretical algorithm from Pantelides with improvements from Soares and Secchi [25] is most commonly used.

Pantelides' approach is to find a minimal structurally singular (MSS) subsets of equations. The equations of the subset are differentiated and replaced by their derivatives. The algebraic variables which get derived with respect to time in the process are marked as states and only their derivatives are considered for the next matching cycle. With the criterion, that the number of new equations generated through differentiation must not exceed the number of variables in the new subset, structural singular systems are detected and the algorithm terminates with an error. Due to the removed algebraic relations between the dynamic variables of the system and the algebraic variables marked as states the calculated results will be unusable. Appropriate algorithms to cover this issue are presented by several authors [16],[18], [19],[20].

3 Matching Algorithms

Since Pantelides' Algorithm does not rely on a particular matching algorithm, it is worth comparing different algorithms within that context. Guided by [9],[14],[24] a set of promising matching algorithms has been selected. While the majority of algorithms is based on a search for augmenting paths, one algorithm employs a push-relabel strategy, designed for maximum flow problems [12],[14]. Since bipartite matching is a special case of the maximum flow problem, push-relabel might be well suited to solve the matching problem [14].

3.1 Augmenting Paths Based Algorithms

3.1.1 DFS

The depth first search based matching algorithm (DFS) applies a depth first search on each unmatched column to find an augmenting path. To avoid double visits an array of size m - the number of rows - is used. The augmenting path can be retrieved from the stack of the DFS. The stack is used to backtrack after visiting all nodes and has the same size as the number of columns n . To improve the performance, an additional array of size n is used to keep the information of the last visited row for each column. In summary the algorithm needs $2n + m$ additional space to the memory for storing the assignments. Please note, that only the Adjacency Matrix but not its transpose is required, since the algorithm traverses only from columns to rows.

3.1.2 BFS

The breadth first search based matching algorithms (BFS) use a breadth first search for each unmatched column to find an augmenting path. The additional space consumption of a good implementation is $n + 2m$. A queue of size n is needed to store the columns to visit next as well as an array of size m to mark the visited rows. The augmenting path is stored in an additional array of size m , saving the parent column to each row. Analogous to the DFS only the Adjacency Matrix is need for BFSB.

3.1.3 MC21A

The MC21A algorithm is based on a DFS with an additional look ahead mechanism. The look ahead mechanism first checks all rows of a column for an unmatched variable before going deeper. Implementing

the look ahead mechanism requires an additional array of size n for the check. In total the implementation needs $3n + m$ additional space.[4][7]

3.1.4 PF

The algorithm by Pothen and Fan (PF) is very much alike MC21A. The difference lies in the usage of the visited flag. A PF phases starts with a queue of size n of all unmatched columns. On each column a DFS with look ahead is applied. The flag *visited* is not reset after the search. The column is dequeued if it is matched. The PF phases are applied until all columns are removed from the queue. The additional space is $4n + m$ and again only the Adjacency Matrix is need for PF.[23]

3.1.5 PF+

PF+ is a simple extension to PF by [9]. To decrease the sensitiveness of the algorithm for row and column permutations the traversal direction of the rows alternates. The additional space consumption is $4n + m$ as in PF.[9][14]

3.1.6 HK

The algorithm by Hopcroft and Karp (HK) is organized in phases comprising two parts. The first part is a BFS from all unmatched columns to assign level numbers to the rows. The level numbers indicate the shortest path length from a row to an unmatched column. In the second part the level numbers are used to increase the assignments with a DFS. It is only allowed to traverse columns with decreasing level numbers. The additional space consumption is $2n + 2m$ (stack(m),queue(n),nextcol(m),levels(m)). Note, since HK uses both BFS and DFS both the Adjacency Matrix and its transposed are required.[13][3]

3.1.7 HKDW

HK modified by Duff and Wiberg (HKDW) adds a third part to the HK phase. The third part is a DFS in the full graph for each remaining unmatched row to increase the matching. The flag *visited* is not reset between two DFS in part three. The additional space consumption with $2n + 2m$ is similar to HK because the additional DFS needs no further memory. [8]

3.1.8 ABMP

The algorithm by Alt et al. (ABMP) is organized in two phases. The first phase increases the matching by a sophisticated search procedure combining BFS and DFS. This phase is performed until the lower bound on the shortest augmenting path length exceeds a suitable value. Alt et al. suggest to use the bound $L = \sqrt{\tau \log n/n}$. [9] The additional space consumption is $2n + 2m$. [1]

3.2 Push Relabel Based Algorithms

Push Relabel Algorithms are developed to solve the problem of maximum flow in networks. The idea behind is not to find augmenting paths but to search and augment together. Based on a set of rules speculative augmentations are performed by unmatching and matching. [14][24]

3.2.1 PR

A detailed description of the implemented push relabel algorithm can be found in [14]. The algorithm uses the same mechanism like PF+ to traverse the adjacency list in alternating order called fairness. The push order to select active columns for pushing is first-in-first-out (FIFO). The additional space consumption is $2n + m$ (row label(m),column label(m),queue(m)) and the Adjacency Matrix as well as its transposed are need.

3.3 Heuristic Based Algorithms

Next to the systematic algorithms discussed above, there are algorithms based on heuristics which are designed to increase the performance of a matching process. They are called cheap matching and their benefits strongly depend on the structure of the problem. Thus they are used as an initial guess or jump-start. In [9] a comprehensive overview on cheap matching algorithms is given. Based on the results from [9] two heuristics are selected for testing. The frequently used and the best one.

3.3.1 Cheap Matching

The cheap matching algorithm traverses all columns and matches the first unmatched row in the adjacency list of the column. The complexity of the algorithm is $O(n + \tau)$.

3.3.2 KS Rand Cheap Matching

The cheap matching algorithm by Karp and Sipser introduces a heuristic based on constructing a smaller graph through two rules and a random matching. More information can be found in [9].

3.4 Adaptability for Index Reduction

The matching algorithms discussed above can be classified based on their behaviour when encountering singular systems. While the *simple* matching algorithms terminate as soon as a single node cannot be assigned, the advanced algorithms terminate with a non empty set of unassigned nodes. Some of them allow the set to be collected in a post processing step.

- Simple Matching Algorithms
 - DFSB
 - BFSB
 - MC21A
- Advanced Matching Algorithms
 - PF
 - PF+
 - HK
 - HKDW
 - ABMP
 - PR

In the original paper of Pantelides, the matching algorithm MC21A by Duff was used. MC21A belongs to the group of simple algorithms. Hence no changes have to be made to the Pantelides Algorithms for simple matching algorithms.

In case of a simple matching algorithm the MSS subset contains exactly one unmatched equation. The other equations of the subset are found by a search in the matched graph starting from the variables of the unmatched equation. During the search, each variable is visited only once. For all presented simple algorithms the search to get the MSS subset is not an extra step, it is found by storing the visited equations in each phase of the algorithm.

In case of an advanced matching algorithm, a search in the matched graph is necessary for each equation to get the MSS subsets. Each subset has to fulfil the criterion, that the number of new equations generated by differentiation must not exceed the number of variables in the new subset. Hence, obtaining the MSS subset is more costly compared to simple algorithms as the search is an extra step.

4 Measurements on Examples

Since there is no comparison of matching algorithms in the field of Modelica known to the author an extensive survey has been conducted. Therefore each matching algorithm has been implemented into the OpenModelica compiler (OMC)³. In order to be compatible with both simple and advanced matching algorithms the Pantelides index reduction had to be reimplemented modifying the interfaces and the compilation process. Since there exists only little experience about the runtime efficiency and comparability of MetaModelica [22], in which the OMC is written, an external C implementation of freely available matching algorithms [15] has been embedded as well.

The aim of this paper is to compare the computational effort of the matching algorithms with and without index reduction using selected examples. In addition the influences of the programming language and the usage of a cheap matching algorithm are investigated.

All measurements were accomplished using a Windows 7, 64 Bit System with Intel Core i7 860, 2.80 GHz and 8.0 GB RAM.

4.1 Examples

To do an extensive comparison of matching algorithms scalable Modelica models are needed. Since the author is mainly concerned with multi body systems, the following mechanical models will be used:

- chain structure Figure 5 (a)
- tree structure Figure 5 (b)
- grassland structure Figure 5 (c)
- kinematic loops 5 (d)

The models are based on the Modelica.Mechanics.Multibody library (MSL 3.1), the Planar Mechanics Library from DLR⁴ and PyMbs [17]. PyMbs⁵ is a Python based multi body tool to generate the equations of motion from a description similar to Modelica.Mechanics.Multibody. PyMbs generates efficient flat Modelica code which places very low demands on the EOOL compiler. Hence no index reduction step is necessary and one obtains a benchmark for pure matching. The reason to use three

³www.openmodelica.org

⁴<http://www.robotic.de/339>

⁵<http://sourceforge.net/projects/pymbs/>

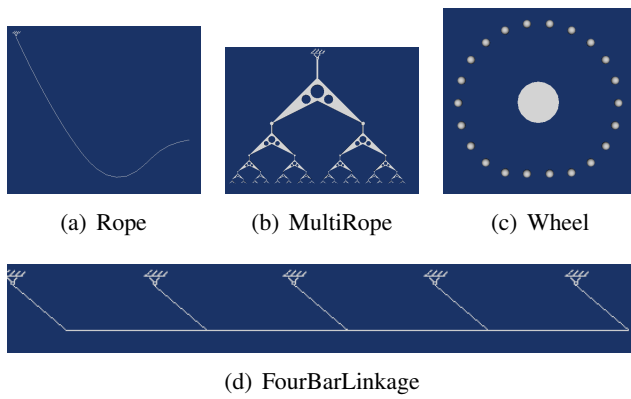


Figure 5: Example Models

different descriptions is to study the influences of the way a model is set up.

In addition to the four models, most examples from the Modelica.Mechanics package are used for the comparison with index reduction.

4.2 Results for Pure Matching

The results for pure matching on the rope model are presented in Figure 6 and Figure 7. Most of the algorithms show a linear relationship between effort and model size. The represented model size is the number of equations the matching algorithm operates on. Note, that this is the reduced size of the model. Because it was important for the benchmarks to be comparable with the usual modelling process all steps, for example the detection of simple equations like $a = b$ and $a = constant$ are performed before matching.

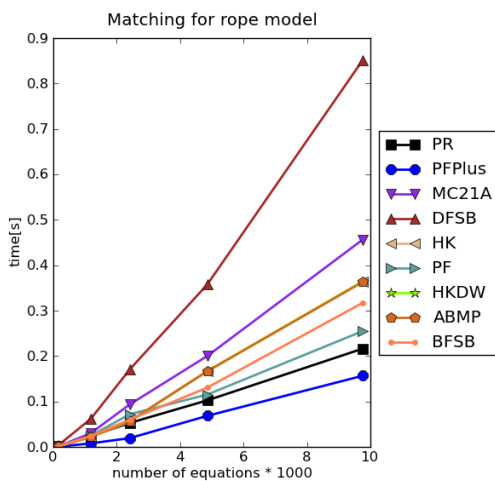


Figure 6: Results from Rope examples, MetaModelica implementation

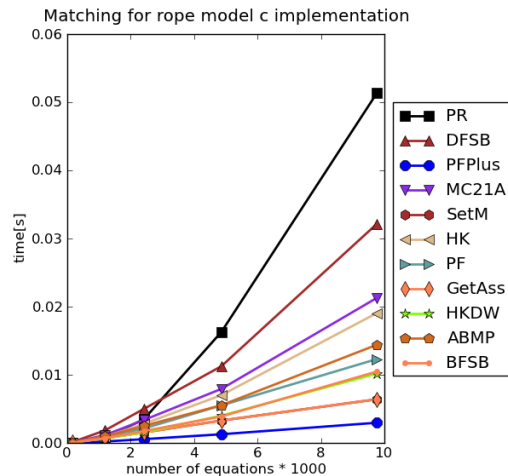


Figure 7: Results from Rope examples, C implementation

The PF+ algorithm is the fastest, while the simple DFS algorithm needs the most time. The PR algorithm is the second fastest, only beaten by PF+. While the MetaModelica implementation suggests that the push relabel algorithm seems to be very efficient, results from the C implementation show a different picture. Here the PR scales non-linear and needs the most time. Again, the DFS is slowest and the PF+ is the fastest augmentation path based algorithm. Generally speaking, the C implementation is around ten times faster than the MetaModelica implementation, including the time to pass the incidence Matrix (SetM) and to return the assignments (GetAss) as shown in Figure 7. Copying the Incidence Matrix and returning the Assignments takes twice the time needed to match the system using the PF+ algorithm, rendering the overall time similar to the fastest MetaModelica implementation. Figure 8 and Figure 9 show the results for the MultiRope model. Again, PF+ is the fastest, DFS needs most time and the C implementation is around 10 times faster.

Figure 10 show the results for the wheel example. Here some algorithms scale non-linear in time and a few scale linear. Still, PF+ is one of the fastest algorithms and DFS needs the most time.

The results for the kinematic loop model are shown in Figure 11. Here, the fastest algorithm is HK closely followed by HKDW. Nonetheless, PF+ still belongs to one of the fastest algorithms.

In summary the fastest overall algorithm in case of pure matching is the PF+ algorithm. It scales linear in time for all test cases and therefore seems well suited for large scale systems.

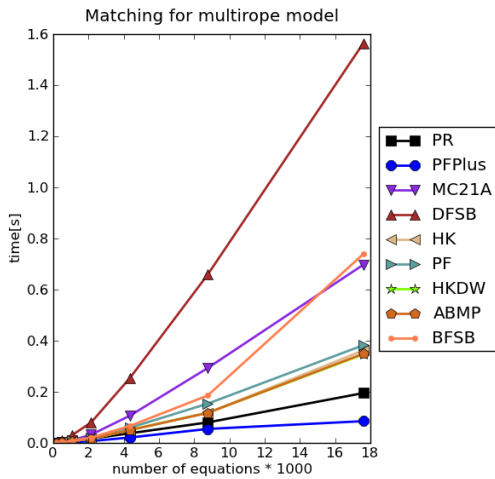


Figure 8: Results from MultiRope examples, MetaModelica implementation

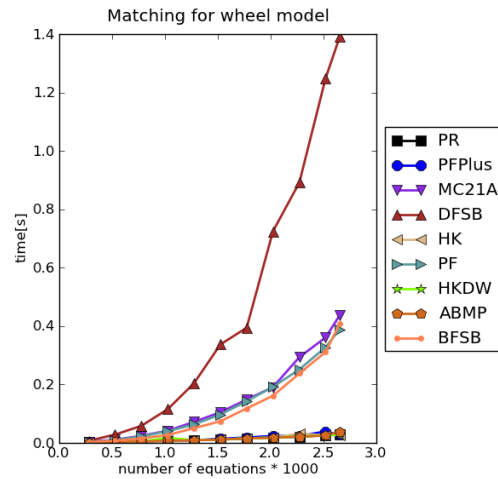


Figure 10: Results from Wheel examples, MetaModelica implementation

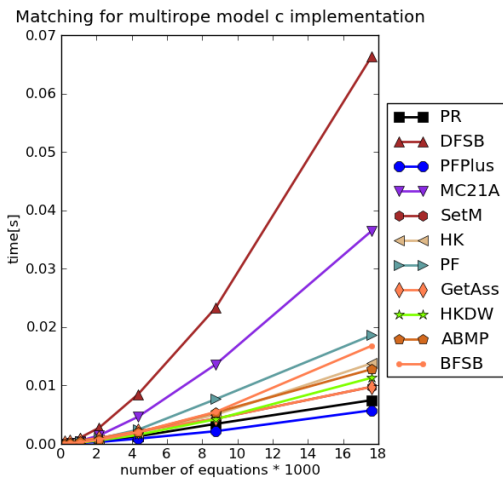


Figure 9: Results from MultiRope examples, C implementation

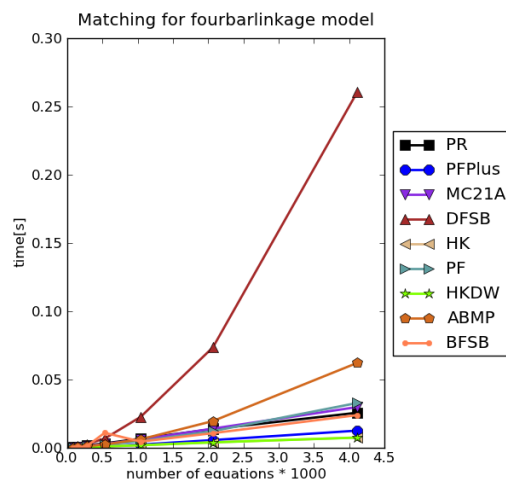


Figure 11: Results from FourBarLinkage examples, MetaModelica implementation

4.3 Results for Matching and Index Reduction

The result for the rope model is shown in Figure 12 and 13. Again PF+ is one of the fastest algorithm and scales linear in time. Since all other models do not show a mentionable difference their results are not shown explicitly. Please note, that due to the lower demands on the EOOL compiler, the OMC manages to process models of up to 200 bodies when described with PyMbs. The upper boundary for the MSL lies at around 50 bodies.

In addition to the models presented above Figure 14 shows the results for the examples included in the package Modelica.Mechanics. The results are presented with a logarithmic time axis. The grey curves

represent linear relationships between time and number of equations. The suffix *Ext* marks the C implementation. Because some models have roughly equal numbers of equations, the graph looks quite scattered. Again, PF+ is one of the fastest algorithm and scales linear in time.

4.4 Results for Cheap Matching

The results from the usage of heuristic algorithms are shown in Figure 15 and 16 for the cheap matching and in Figure 17 and 15 for the KS cheap matching algorithm. It can be seen that especially the BFS and DFS MetaModelica implementations benefit from the usage of a cheap matching algorithm. The time saved

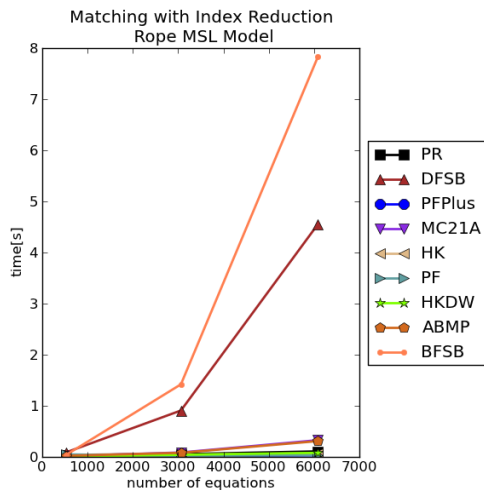


Figure 12: Results from Rope MSL examples, Meta-Modelica implementation

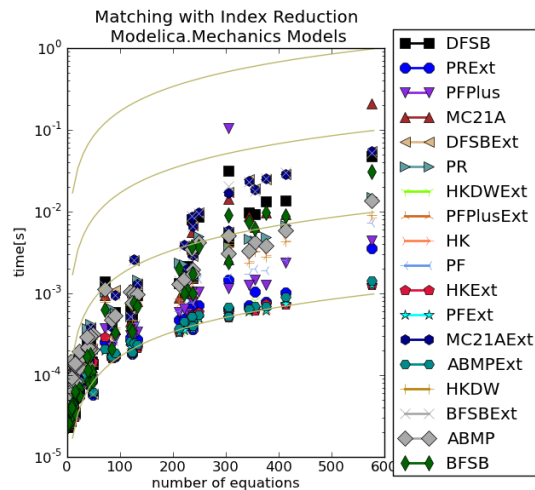


Figure 14: Results from Matching with Index Reduction for Modelica.Mechanics Example Models

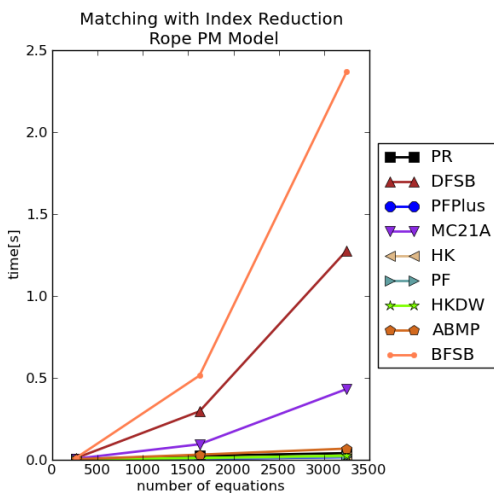


Figure 13: Results from Rope PM examples, Meta-Modelica implementation

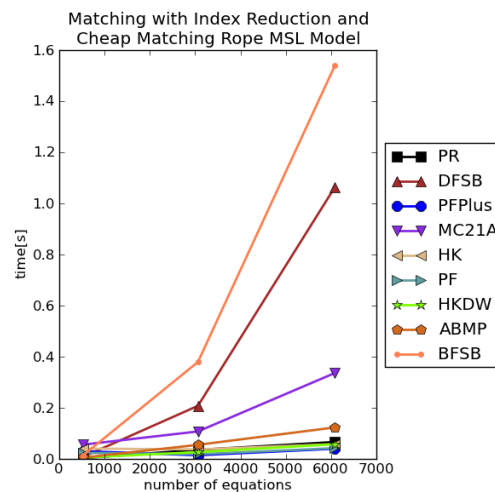


Figure 15: Results from Rope MSL examples, Meta-Modelica implementation

for both algorithm is around 80%.

5 Conclusion

An extensive survey has been conducted by the author to find the best suited matching algorithm for EOOL compilers. Several real life models have been used for testing. It was found that that the PF+ algorithm performed best on almost all models.

Moreover, it has been found that the PF+ algorithm, although it has a non-linear worst case time complexity, scales linear for the models tested within this survey. This makes it ideally suited for the application in large scale models. Unfortunately, further increase in model size, to support that claim, was hindered due to

the memory consumption of the OpenModelica compiler. Future work will aim at increasing the manageable model size and rerun the benchmarks.

It could also be shown that MetaModelica seems not to be well suited for such algorithms since the C implementation is at least 10 times faster. Maybe some further language and compiler features could decrease the time difference to a natural C implementation. The main difference of implementation is caused by the storage of the Adjacency Matrix. The C implementation uses an array to store the values and an additional array to store the column indices. In MetaModelica the matrix is stored as an array of lists. To traverse the lists in MetaModelica recursive function calls are needed whereas the c implementation simply stores

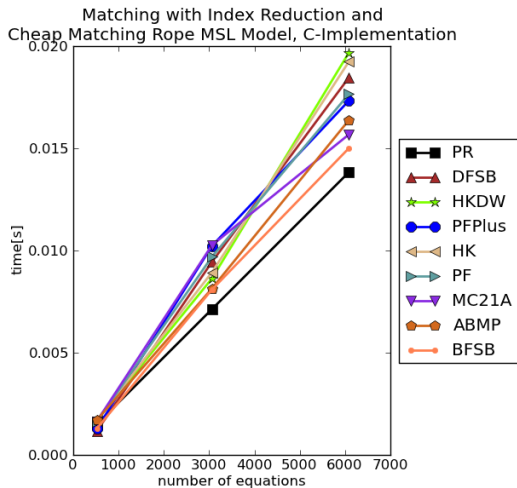


Figure 16: Results from Rope MSL examples, C implementation

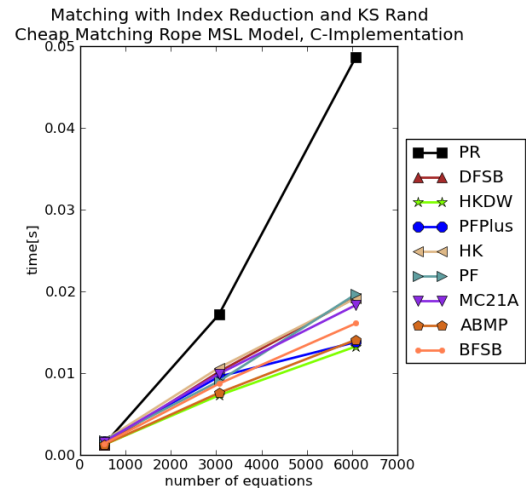


Figure 18: Results from Rope MSL examples, C implementation

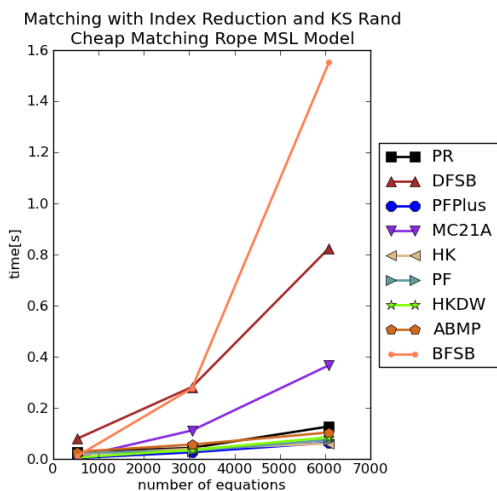


Figure 17: Results from Rope MSL examples, Meta-Modelica implementation

the needed indices for the traversal in arrays.

Since the implementation is freely available in the OpenModelica Compiler, the survey may be extended with models from other physical domains.

References

- [1] Alt, H.; Blum, N.; Mehlhorn, K.; Paul, M.: Computing a maximum cardinality matching in a bipartite graph in time $O(n^{1.5}\sqrt{m/\log(n)})$, Information Processing Letters, Volume 37, Issue 4, 28 February 1991, Pages 237-240
- [2] Berge, C. The Theory of Graphs. Methuen, London, 1962
- [3] Blum, N.: A simplified realization of the Hopcroft-Karp approach to maximum matching in general graphs. Technical report, Universität Bonn, 1999.
- [4] Duff, I. S. On algorithms for obtaining a maximum transversal. ACM Trans. Math. Softw. 7(1981), 315-330.
- [5] Duff, I.S.; Erisman, A.M.; Reid, J.K.: Direct methods for sparse matrices, 1986, Clarendon Press Oxford
- [6] Duff, I. S.; Reid J. K.; Harwell, A.: An implementation of Tarjan's algorithm for the block triangularization of a matrix, in ACM Trans. Math. Software Volume 4, pp. 137-147, 1978
- [7] Duff, I. S.: Algorithm 575: Permutations for a Zero-Free Diagonal [F1]. ACM Trans. Math. Softw. 7, 3 September 1981, 387-390
- [8] Duff, I. S.; Wiberg, T.: Remarks on implementation of $O(n^{1/2}\tau)$ assignment algorithms, in ACM Trans. Math. Software Volume 1 4, pp. 267-287, 1988
- [9] Duff, I.S.; Kaya, K.; Uçar, B.: Design, implementation, and analysis of maximum transversal algorithms, ACM Transactions on Mathematical Software (TOMS), 38, 2, 13, 2011, ACM
- [10] Elmquist, H.: A Structured Model Language for Large Continuous Systems, Ph.D. Dissertation, Report CODEN: LUTFD2/(TFRT-1015), Dept.

- of Automatic Control, Lund Institute of Technology, Lund, Sweden, 1978
- [11] Frenkel, J.; Schubert, C.; Kunze, G.; Fritzson, P.; Sjölund, M.; Pop, A.: Towards a Benchmark Suite for Modelica Compilers: Large Models. In: Proceedings of the 8th Modelica Conference 2011, Dresden, Germany, Modelica Association, 20-22 March 2011. https://www.modelica.org/events/modelica2011/Proceedings/pages/papers/07_1_ID_183_a_fv.pdf
- [12] Goldberg, A. V.; Tarjan, R. E.: A new approach to the maximum flow problem. Annual ACM Symposium on Theory of Computing, Proceedings of the eighteenth annual ACM symposium on Theory of computing, 136-146
- [13] Hopcroft, J. E.; Karp, R. M.: A $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. SIAM Journal of Computing, 2(4): 225-231, 1973
- [14] Kaya, K.; Langguth, J.; Manne, F.; Uçar, B.: Experiments on Push-Relabel-based Maximum Cardinality Matching Algorithms for Bipartite Graphs, CERFACS Tech. Report TR/PA/11/33, May, 2011
- [15] Kaya, K.: <http://bmi.osu.edu/kamer/research.html>, last visit 2012-02-05, Matchmaker v0.3
- [16] Kunkel, P.; Mehrmann, V.: Index reduction for differential-algebraic equations by minimal extension. Z. angew. Math. Mech., 84: pp. 579-597, 2004
- [17] Kunze, G.; Frenkel, J.; Knoll, C.; Schubert C.; Voigt, S.: PyMbs: Ein generisches Software Werkzeug für die Simulation von Mehrkörper-systemen, VDI Mechatronik Tagung, 2011.
- [18] Mattsson, S.; Söderlind, G.: Index reduction in differential-Algebraic equations using dummy derivatives, SIAM J. Sci. Comput. 14, 677-692, 1993.
- [19] Mattsson, S.E.; Olsson, H; Elmqvist, H. Dynamic Selection of States in Dymola. In: Proceedings of the Modelica Workshop 2000, Lund, Sweden, Modelica Association, 23-24 Oct. 2000.
- [20] Mattsson, S.E.; Söderlind, G.: A new technique for solving high-index differential-algebraic equations using dummy derivatives, Computer-Aided Control System Design, 1992. (CACSD), 1992 IEEE Symposium on , pp.218-224, 17-19 Mar 1992
- [21] Pantelides C. The Consistent Initialization of Differential-Algebraic Systems. SIAM J. Sci. and Stat. Comput. Volume 9, Issue 2, pp. 213-231, March 1988.
- [22] Pop, A.; Fritzson, P.: MetaModelica: A Unified Equation-Based Semantical and Mathematical Modelling Language. In Proceedings of Joint Modular Languages Conference 2006 (JMLC2006) LNCS Springer Verlag. Jesus College, Oxford, England, Sept 13-15, 2006.
- [23] Pothén, A; Fan; C.-J.: Computing the block triangular form of a sparse matrix. ACM Trans. Math. Softw. 16, 4 ,December 1990, 303-324
- [24] Setubal, J.C.: Sequential and parallel experimental results with bipartite matching algorithms , in Technical Report EC-96-09, Institute of Computing, University of Campinas, Brasil, 1996
- [25] Soares, R. de P.; Secchi, A. R.: Direct Initialisation and Solution of High-Index DAE Systems. in Proceedings of the European Symposium on Computer Aided Process Engineering - 15, Barcelona, Spain, 2005
- [26] R. Tarjan, Depth-first search and linear graph algorithms, in Conf. Record 1971 IEEE 12th Annu. Symp. Switch. Automata Theory, 1971, pp. 114-121
- [27] Unger, J.; Kröner, A.; Marquardt, W.: Structural analysis of differential-algebraic equation systems-theory and applications, Computers & Chemical Engineering, Volume 19, Issue 8, August 1995, Pages 867-882

Static and Dynamic Debugging of Modelica Models

Adrian Pop¹, Martin Sjölund¹, Adeel Asghar¹, Peter Fritzson¹, Francesco Casella²

¹Programming Environments Laboratory

Department of Computer and Information Science

Linköping University, Linköping, Sweden

²Dipartimento di Elettronica e Informazione, Politecnico di Milano, Milano, Italy

{adrian.pop,martin.sjolund,adeel.asghar,peter.fritzson}@liu.se

casella@elet.polimi.it

Abstract

The high abstraction level of equation-based object-oriented languages (EOO) such as Modelica has the drawback that programming and modeling errors are often hard to find. In this paper we present static and dynamic debugging methods for Modelica models and a debugger prototype that addresses several of those problems. The goal is an integrated debugging framework that combines classical debugging techniques with special techniques for equation-based languages partly based on graph visualization and interaction.

To our knowledge, this is the first Modelica debugger that supports both transformational and algorithmic code debugging.

Keywords: Modelica, Debugging, Modeling and Simulation, Transformations, Equations, Algorithmic Code, Eclipse

1 Introduction

Advanced development of today's complex products requires integrated environments and equation-based object-oriented declarative (EOO) languages such as Modelica [8][12] for modeling and simulation. The increased ease of use, the high abstraction, and the expressivity of such languages are very attractive properties. However, these attractive properties come with the drawback that programming and modeling errors are often hard to find.

To address these issues we present static (compile-time) and dynamic (run-time) debugging methods for Modelica models and a debugger prototype that addresses several of those problems. The goal is an integrated debugging framework that combines classical debugging techniques with special techniques for equation-based languages partly based on graph visualization and interaction.

The static transformational debugging functionality addresses the problem that model compilers are optimized so heavily that it is hard to tell the origin of an equation during runtime. This work proposes and implements a prototype of a method that is efficient with less than one percent overhead, yet manages to keep track of all the transformations/operations that the compiler performs on the model.

Modelica models often contain functions and algorithm sections with algorithmic code. The fraction of algorithmic code is increasing since Modelica, in addition to equation-based modeling, is also used for embedded system control code as well as symbolic model transformations in applications using the MetaModelica language extension.

Our earlier work in debuggers for the algorithmic subset of Modelica used high-level code instrumentation techniques which are portable but turned out to have too much overhead for large applications. The new dynamic algorithmic code debugger is the first Modelica debugger that can operate without high-level code instrumentation. Instead, it communicates with a low-level C-language symbolic debugger to directly extract information from a running executable, set and remove breakpoints, etc. This is made possible by the new bootstrapped OpenModelica compiler which keeps track of a detailed mapping from the high level Modelica code down to the generated C code compiled to machine code.

The dynamic algorithmic code debugger is operational, supports both standard Modelica data structures and tree/list data structures, and operates efficiently on large applications such as the OpenModelica compiler with more than 100 000 lines of code.

The attractive properties of high-level object-oriented equation-based languages come with the drawback that programming and modeling errors are often hard to find. For example, in order to simulate models efficiently, Modelica simulation tools perform a large number of symbolic manipulation in order to

reduce the complexity of models and prepare them for efficient simulation. By removing redundancy, the generation of simulation code and the simulation itself can be sped up significantly. The cost of this performance gain is error-messages that are not very user-friendly due to symbolic manipulation, renaming and reordering of variables and equations. For example, the following error message says nothing about the variables involved or its origin:

```
Error solving nonlinear system 2
time = 0.002
residual[0] = 0.288956, x[0] = 1.105149
residual[1] = 17.000400, x[1] = 1.248448
```

It is usually hard for a typical user of the Modelica tool to determine what symbolic manipulations have been performed and why. If the tool only emits a binary executable this is almost impossible. Even if the tool emits source code in some programming language (typically C), it is still quite hard to know what kind of equation system you have ended up with. This makes it difficult to understand where the model can be changed in order to improve the speed or stability of the simulation. Some tools allow the user to export the description of the translated system of equations [18], but this is not enough. After symbolic manipulation, the resulting equations no longer need to contain the same variables or structure as the original equations.

This work proposes and develops a combination of static and dynamic debugging techniques to address these problems. The static (compile-time) transformational debugging efficiently traces the symbolic transformations throughout the model compilation process and provides explanations regarding to origin of problematic code. The dynamic (run-time) debugging allows interactive inspection of large executable models, stepping through algorithmic parts of the models, setting breakpoints, inspecting and modifying data structures and the execution stack.

An integrated approach is proposed where the origin mapping provided by the static transformational debugging is used by the dynamic debugger to relate run-time errors to the original model sources. To our knowledge no other open-source or commercial Modelica tool currently supports static transformational debugging or algorithmic code debugging.

The paper is structured as follows: Section 2 the background and related work, Section 3 analyzes sources of errors and faults, Section 4 proposes an integrated static and dynamic debugging approach, Section 5 presents the static transformational debugging method and implementation, whereas Section 6 presents the algorithmic code debugging functionality. Conclusions and future work are given in Section 7.

2 Background and Related Work

2.1 Debugging techniques for EOO Languages

In the context of debugging declarative equation-based object-oriented (EOO) languages such as Modelica, both the static (compile-time) and the dynamic (run-time) aspects have to be addressed.

The static aspect of debugging EOO languages deals with inconsistencies in the underlying system of equations:

1. Errors related to the *transformations* of the models to an optimized flattened system of equations suitable for numeric solution, e.g. symbolic solutions leading to division by a constant zero stemming from a singular system of equations, or (very rarely) errors in the symbolic transformations themselves.
2. Overconstrained models (too many equations) or underconstrained models (too few equations). The number of variables needs to be equal to the equations is required for solution.

The dynamic (run-time) aspect of debugging EOO languages addresses run-time errors that may appear due to faults in the model:

1. *model configuration*: when the parameters values and start attributes for the model simulation are incorrect.
2. *model specification*: when the equations and algorithm sections that specify the model behavior are incorrect.
3. *algorithmic code*: when the functions called from equations return incorrect results.

Methods for both static and dynamic (run-time) debugging of EOO languages such as Modelica have been proposed earlier [6][7]. With the new Modelica 3.0 language specification, the static overconstrained/underconstrained debugging of Modelica presents a rather small benefit, since all models are required to be balanced. All models from already checked libraries will already be balanced; only newly written models might be unbalanced, which is particularly useful if new models contain a significant number of unknowns.

Regarding dynamic (run-time) debugging of models [6] proposes a semi-automated declarative debugging solution in which the user has to provide a correct diagnostic specification of the model which is used to generate assertions at runtime. Moreover, starting from an erroneous variable value the user explores the dependent equations (a slice of the program) and acts like an “oracle” to guide the debugger in finding the error.

3 Sources of Errors and Faults

There are a number of sources of errors and faults in a simulation system. Some errors can be recovered automatically by the system, whereas others should be reported and allow the users to enter debugging mode. An error can also be a wrong value pointed out manually by a user.

Every solver employed within a simulation system at all levels should be equipped with an error reporting mechanism, allowing error recovery by the master solver, or error reporting to the end-user in case of irrecoverable error:

- the ODE solvers
- the functions computing the derivatives and the algebraic functions given the states, time, and inputs
- the functions computing the initial states and the values of parameters
- the linear equation solvers
- the nonlinear equation solvers

If some equation can be solved symbolically, without resorting to numerical solvers, then the symbolic solution code should be equipped with diagnostics to handle errors as well.

In the next section we give causes of errors that can appear during the model simulation.

3.1 Errors in the evaluation of expressions

During the evaluation of expressions, faults may occur due to the following causes:

- Division by zero
- Evaluation of non-integer powers with negative argument
- Functions called outside their domain (e.g.: $\text{sqrt}(-1)$, $\text{log}(-3)$, $\text{asin}(2)$). For non built-in functions, these errors can be triggered by assertions within the algorithm, or by calls to the pre-defined `ModelicaError()` function in the body of external functions.
- Errors manifesting as computed wrong value of some variable(s), where the error is manually pointed out by a user or automatically detected as being outside min/max bounds.

3.2 Assertion violations in models

During initialization or simulation, assertions inside models can be triggered when the condition being asserted becomes false.

3.3 Errors in the solution of implicit algebraic equations

During initialization or simulation of DAE systems, implicit equations (or systems of implicit equations, corresponding to strong components in the BLT decomposition) must be solved. In the case of linear systems, the solver might fail because there is some error in evaluating the coefficients of the A matrix and of the b vector of the linear equation $Ax = b$, or because said problem is singular. In the case of nonlinear equations $f(x) = 0$, the solver might fail for several reasons: the evaluation of the residual $f(x)$ or of its Jacobian gives errors; the Jacobian becomes singular: the solver fails to converge after a maximum number of iterations.

3.4 Errors in the integration of the ODEs

In OpenModelica, the DAEs are brought to index-1 ODE form by symbolic and numerical transformation, and these equations are then solved by an ODE solver, which iteratively computes the next state given the current state. During the computation of the next state, e.g. by using Euler, Runge-Kutta or a BDF algorithm, errors such as those reported in section 3.1, 3.2, 3.3 might occur. Furthermore, the solver might fail because of singularity in the ODE, as in the case of finite escape time solutions, or of discontinuities leading to chattering.

4 Integrated Debugging Approach

In this section we propose an integrated debugging method combining information from a static analysis of the model with dynamic debugging at run-time.

4.1 Integrated Static-Dynamic Debug Method

This method partly follows the approach proposed in [6][7] and further elaborated in [3]. However, our approach does not require the user to write diagnostic specifications of models. Also, the approach we present here can also handle the debugging of algorithmic code using classic debugging techniques.

An overview of this debugging strategy is presented in Figure 1. In short, our run-time debugging method is based on the integration of the following:

1. Dependency graph visualization and interaction.
2. Presentation of simulation results and modeling code.
3. Mapping of errors to model code positions.
4. Execution-based debugging of algorithmic code.

A possible debugging session might be as follows.

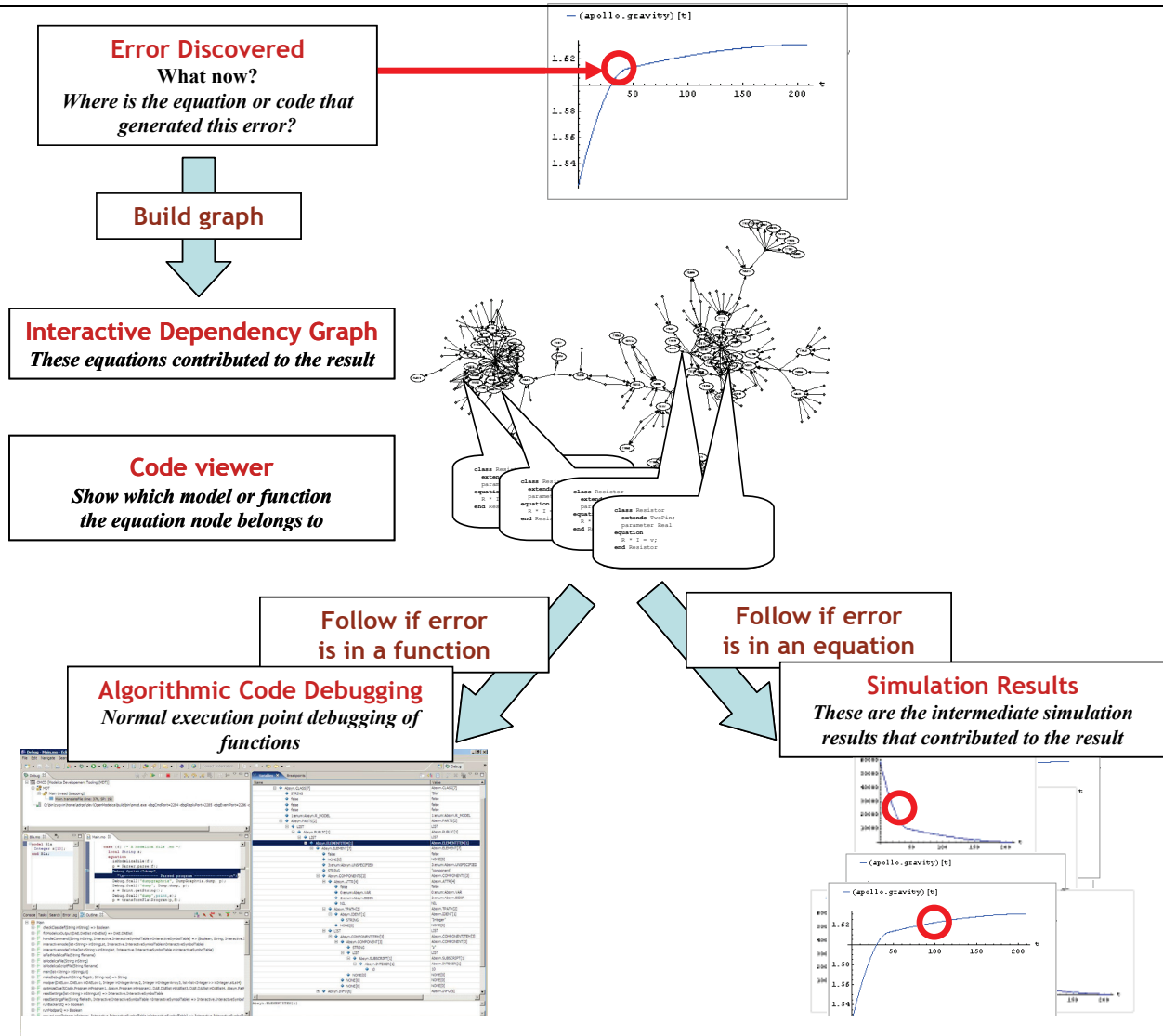


Figure 1. Integrated debugging approach overview.

During the simulation phase, the user discovers an error in the plotted results, or an irrecoverable error is triggered by the run-time simulation code. In the former case, the user marks either the entire plot of the variable that presents the error or parts of it and starts the debugging framework. The debugger presents an (IDG) interactive dependency graph with respect to the variable with the wrong value or the expression where the fault occurred. The dependency edges in IDG are computed using the transformation tracing that is described in Section 5. The nodes in the graph consist of all the equations, functions, parameter value definitions, and inputs that were used to calculate the wrong variable value, starting from the known values of states, parameters and time.

The variable with the erroneous value (or which cannot be computed at all) is displayed in a special node which is the root of the graph. The IDG contains two types of edges:

1. *Calculation dependency edges*: the directed edges labeled by variables or parameters which are inputs

(used for calculations in this equation) or outputs (calculated from this equation) from/to the equation displayed in the node.

2. *Origin edges*: the undirected edges that tie the equation node to the actual model which this equation belongs to.

The user interacts with the dependency graph in several ways:

- *Displaying simulation results* through selection of the variables (or parameters) names (edge labels). The plot of a variable is shown in a popup window. In this way the user can quickly see if the plotted variable has erroneous values.
- *Displaying model code* by following origin edges.
- *Invoking the algorithmic code debugging subsystem* when the user suspects that the result of a variable calculated in an equation which contains a function call is wrong, but the equation seems to be correct.

Using these interactive dependency graph facilities the user can follow the error from its manifestation to its origin. Note that in most cases of irrecoverable errors

arising when trying to compute a variable, the root cause of the error does not lie in the equation itself being wrong, but rather in some of the values of previously computed variables appearing in it being wrong, e.g., because of erroneous initialization or parameterization.

The proposed debugging method can also start from multiple variables with wrong values with the premise that the error might be at the confluence of several dependency graphs.

Note that the debugger can handle both data dependency edges (e.g. which variables influence the current variable of interest), and origin edges (edges pointing from the generated executable simulation code to the original equations/parts of equations contributing to this code). Both are computed by the transformational debugger mentioned in Section 5.

5 Static Transformational Debugging

Transformational debugging is a static compile-time technique since it does not need run-time execution of a model. The method keeps track of symbolic transformations, can explain and display applied transformations, and compute dependence edges between the original model and the generated executable code.

5.1 Common Operations on Continuous Equation Systems

In order to create a debugger adapted for debugging the symbolic transformations performed on equation systems, its requirements should be stated. There are many symbolic operations that may be performed on equation systems. The following descriptions of operations also include a rationale for each of them, since it is not always apparent why perform certain operations are performed. There are of course many more operations that can be performed than the ones listed below, which are however deemed most important, and which the debugger for models translated by the OpenModelica Compiler [11] should be able to handle.

5.1.1 Variable aliasing

An optimization that is very common in Modelica compilers is variable aliasing. This is due to the connection semantics of the Modelica language. For example, if a and b are connectors with the effort-variable v and flow-variable i , a connection (2) will generate alias equations (3) and (4).

$$\text{connect}(a, b) \quad (2)$$

$$a.v = b.v \quad (3)$$

$$a.i + b.i = 0 \Rightarrow b.i = -a.i \quad (4)$$

In a result-file, this alias relation can be stored instead of a duplicate trajectory, saving both space and computation time. In the equation system, $b.v$ may be substituted by $a.v$ and $b.i$ by $-a.v$, which may lead to further optimizations of the equations.

5.1.2 Known variables

Known variables are similar to alias variables in that you may perform variable substitutions on the rest of the equation system if you find such an occurrence. For example, (5) and (6) can be combined into (7). In the result-file, you no longer need to store a value for each time step; once is enough for known variables (which have values that can be computed statically at compile-time), parameters and constants.

$$a = 4.0 \quad (5)$$

$$b = 4.0 - a + c \quad (6)$$

$$b = 4.0 - 4.0 + c \quad (7)$$

5.1.3 Equation Solving

If the tool has determined that x needs to be solved for in (8), we need to symbolically solve the equation, producing a simple equation with x on one side as in (9). Solving for x is not always straightforward, and it is not always possible to invert user-defined functions such as (10). Since x is present in the call arguments and the function cannot be inverted or inlined, it is not possible to solve the equation symbolically, so it is necessary to resort to a numerical non-linear solver during runtime.

$$15.0 = 3.0 * (x + y) \quad (8)$$

$$x = 15.0 / 3.0 - y \quad (9)$$

$$0 = f(3 * x) \quad (10)$$

5.1.4 Expression Simplification

Expression simplification is a symbolic operation that does not change the meaning of the expression, while making it faster to calculate. It is related to many different optimization techniques such as constant folding. The order in which arguments are evaluated may be changed (11). Constant subexpressions are evaluated during compile-time (12). Non-constant subexpressions may be rewritten (13) and functions may be evaluated fewer times than in the original expression (14). It is also possible to use special knowledge about an expression in order to make it run faster (15) and (16).

$$\text{and}(a, \text{false}, b) \Rightarrow \text{false} \quad (11)$$

$$4.0 - 4.0 + c \Rightarrow c \quad (12)$$

$$\text{max}(a, b, 7.5, a, 15.0) \Rightarrow \text{max}(a, b, 15, 0) \quad (13)$$

$$f(x) + f(x) + f(x) \Rightarrow 3 * f(x) \quad (14)$$

$$\text{if cond then } a \text{ else } a \Rightarrow a \quad (15)$$

$$\text{if cond then false else true} \Rightarrow \text{cond} \quad (16)$$

5.1.5 Equation System Simplification

It is of course also possible to solve some equation systems statically. For example a linear system of equations with constant coefficients (17) can be solved using one step of symbolic Gaussian elimination (18), generating two separate equations that can be solved individually after causalization (19). A simple linear equation system as (17) may also be solved numerically using e.g. LAPACK [1] routines.

$$\begin{bmatrix} 1, & 2; & 2, & 1 \end{bmatrix} * \begin{bmatrix} x; & y \end{bmatrix} = \begin{bmatrix} 4; & 5 \end{bmatrix} \quad (17)$$

$$\begin{bmatrix} 1, & 2; & 0, & -3 \end{bmatrix} * \begin{bmatrix} x; & y \end{bmatrix} = \begin{bmatrix} 4; & -3 \end{bmatrix} \quad (18)$$

$$x = 2; \quad y = 1; \quad (19)$$

5.1.6 Differentiation

Symbolic differentiation [16] is used for many purposes. It is used to expand known derivatives (20) or as one operation in index reduction. Jacobian matrices have many applications, e.g. to speed up simulation runtime [14]. The matrix is often computed using automatic differentiation [14][16] which combines symbolic differentiation with other techniques to achieve fast computation.

$$\text{der}(t^2, t) = 2*t \quad (20)$$

5.1.7 Index reduction

In order to solve DAE's numerically, discretization techniques and methods to numerically compute derivatives are used (often referred to as solvers). Certain DAE's need to be differentiated symbolically to enable a stable numeric solution. The differential index of a general DAE system is the minimum number of times that certain equations in the system need to be differentiated to reduce the system to a set of ODEs, which can then be solved by the usual ODE solvers, Chapter 18 in [8]. While there are techniques to solve DAE's of higher index than 1, most of them require index-1 DAE's (no second derivatives). This makes it more convenient to reformulate the problem using index reduction algorithms, Chapter 18 in [8]. One such technique uses dummy derivatives [15]; this is the algorithm currently used in the OpenModelica Compiler.

5.1.8 Function inlining

Writing functions to do common operations is a great way to reduce the burden of maintaining code. When a function call is inlined (21), it can be treated as a macro expansion (22) and may increase the number of symbolical manipulations that can be perform on an expression such as (23).

$$2*f(x, y)/\pi \quad (21)$$

$$2*\pi*((\sin(x+y)+\cos(x+y-y))/\pi) \quad (22)$$

$$2*(\sin(x+y) + \cos(x)) \quad (23)$$

5.2 Debugging

The choice of techniques for implementation of a debugger depends on where and for what it is intended to be used. Translation and optimization of large application models can be very time-consuming. Thus it would be good if the approach has such a low overhead that it can be enabled by default. It would also be good if error messages from the runtime could use the debug information from the translation and optimization stages to give more understandable and informative messages to the user.

A technique that is commonly used for debugging is tracing. The simplest way of implementing tracing is to print a message to the terminal or file in order to log the operations that you perform. The problem here is that if an operation is rolled back, the log-file will still contain the operation that was rolled back. The data also need to be post-processed if the operations should be grouped by equation.

A more elegant technique is to treat operations as metadata on equations, variables or equation systems. Other metadata that should already be propagated from source code to runtime include the name of the component that an equation is part of, which line and column that the equation originates from, and more. Whenever an operation is performed, the operation kind and input/output is stored inside the equation as a list of operations. If the structure used to store equations is persistent this also works if the tool needs to roll back execution to an earlier state.

The cost of adding this meta data is a constant runtime factor from storing a new head in the list. The memory cost depends a lot on the compiler itself. If garbage collection or reference counting is used, the only cost is a small amount to describe the operation (typically an integer and some pointers to the expressions involved in the operation).

5.3 Bookkeeping of Operations

5.3.1 Variable Substitution

The elimination of variable aliasing and variables with known values (constants) is considered as the same operation that can be done in a single phase. It can be performed as a fixed-point algorithm where substitutions are collected which record if any change was made (stop if no substitution is performed or no new substitution can be collected). For each alias or known variable, merge the operations stored in the simple equation $x = y$ before removing it from the equation system. For each successful substitution, record it in the list of operations for the equation.

The history of the variable a in the equation system (24) could be represented as a more detailed version (25) instead of the shorter (26) depending on the order in which the substitutions were performed.

$$a = b; b = -c; c = 4.5 \quad (24)$$

$$a = b \Rightarrow a = -c \Rightarrow a = -4:5 \quad (25)$$

$$a = b \Rightarrow a = -4.5 \quad (26)$$

In equation systems that originate from a Modelica model it is preferable to see a substitution as a single operation rather than a longer chain of operations (chains of 50 cascading substitutions are not unheard of and makes it hard to get an overview of the operations performed on the equation, even though sometimes all the steps are necessary to understand the reason for the final substitution).

It is also possible to collect sets of aliases and select a single variable (doing everything in one operation) in order to make substitutions more efficient. However, alias elimination may still cascade due to simplification rules (27), which means that you need a work-around for substitutions performed in a non-optimal order.

$$\begin{aligned} a = b - c + d &\Rightarrow a = b - b + d \\ &\Rightarrow a = d \end{aligned} \quad (27)$$

Thus, we compare the previous operation with the new one and if we detect a link in the chain, we store this relation. When displaying the operations of an equation system, it is then possible to expand and collapse the chain depending on the user's needs.

5.3.2 Equation Solving

Some equations are only valid for a certain range of input. When solving an equation like (28), you assert that the divisor is non-zero and eliminate it in order to solve for x . We record a list of the assertions made (and their sources for traceability). An assertion may be removed if we later determine that it always holds or if it overlaps with another assertion (29).

$$x/y = 1 \Rightarrow x = y \quad (y \neq 0) \quad (28)$$

$$y \neq 0, 4.0 < y < 8.0 \Rightarrow 4.0 < y < 8.0 \quad (29)$$

5.3.3 Expression Simplification

Tracking changes to an expression is easy if you have a working fixed-point algorithm for expression simplification (record a simplification operation if the simplification algorithm says that the expression changed). However, if the simplification algorithm oscillates (as in 30) it is hard to use it as a fixed-point algorithm.

$$2*x \Rightarrow x*2 \Rightarrow 2*x \Rightarrow \dots \quad (30)$$

The simple solution is to use an algorithm that is fixed point, or conservative (reporting no change made when

performing changes that may cause oscillating behavior). Finding where this behavior occurs is not hard for a compiler developer (simply print an error message after 10 iterations). If it is hard to detect if a change has actually occurred (due to changing data representation to use more advanced techniques), one may need to compare the input and output expression in order to determine if the operation should be recorded. While comparing large expressions may be expensive, it is often possible to let the simplification routine keep track of any changes at a smaller cost.

5.3.4 Equation System Simplification

It is possible to store these operations as pointers to a shared and more global operation or as many individual copies of the same operation. It is preferable to store this as a single global operation (see Figure 2) since the only cost is only some indirection when reading the data. It is also recommended to store reverse pointers (or indices) from the global operation back to each individual operation as well, so that reverse lookup can be performed at a low cost.

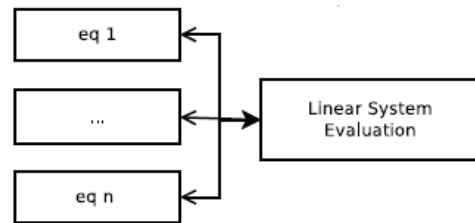


Figure 2. Sharing Results of Linear System Evaluation.

As the tool we are using performs only limited simplification of these strongly connected components, we are currently limited to only recording evaluation of constant linear systems. As more of these optimizations are added to the compiler, they will also need to be traced and support added for them in the debugger.

5.3.5 Differentiation

Whenever we perform symbolic differentiation in an expression, e.g. to expand known derivatives (31), we record this operation in the equation. OpenModelica currently does not eliminate this state variable as in (32), but if it did the operation would also be recorded.

$$\text{der}(x) = \text{der}(\text{time}) \Rightarrow \text{der}(x) = 1.0 \quad (31)$$

$$\begin{aligned} \text{der}(x) = 1.0 &\Rightarrow \\ x &= \text{time} + (x_{\text{start}} - \text{timestart}) \end{aligned} \quad (32)$$

5.3.6 Index reduction

For the index reduction algorithm, any performed substitution is recorded, source information is added to the newly introduced dummy derivative variable, and the

operations are performed on the affected equations. As an example for the dummy derivatives algorithm, this includes differentiation of the Cartesian coordinates $(x; y)$ of a pendulum with length L (33) into (34) and (35). After the index reduction is complete, further optimizations such as variable substitution (37), are performed to reduce the complexity of the complete system.

$$x^2 + y^2 = L^2 \tag{33}$$

$$\text{der}(x^2 + y^2) \Rightarrow 2 * (\text{der}(x) * x + \text{der}(y) * y) \tag{34}$$

$$\text{der}(L^2) \Rightarrow 0 \tag{35}$$

$$2 * (\text{der}(x) * x + \text{der}(y) * y) \Rightarrow 2 * (u * x + v * y) \tag{36}$$

5.3.7 Function inlining

Since inlining functions may cause a new function call to be added to the expression, functions are inlined until a fixed point is reached (with a maximum depth to avoid problems with recursive functions). Expressions are also simplified in order to reduce the size of the final expression. When inlining calls in an equation have been completed, this is recorded as an inline operation with the expression before and after.

5.4 Presentation of Operations

Until now the focus has been on collecting operations as data structured in the equation system. What is it possible to do with this information? During the translation phase, it can be used directly to present information to the user. Assuming that the data is well structured, it is possible to store it in a static database (e.g. SQL) or simply as structured data (e.g. XML). That way the data can be accessed by various applications and presented in different ways according to the user needs for all of them. The current OpenModelica prototype only outputs text at present; in the future this information will be presented in the origin edge introduced in Section 4.

The number of operations stored for each equation varies widely. The reason is that when a known variable x is replaced with, e.g., the number 0.0, one may start removing subexpressions. One then ends up with a chain of operations that loops over variable substitutions and expression simplification. The number of operations performed may scale with the total number of variables in the equation system if the the number of iterations that the optimizer may take is not limited [17]. This makes some synthetic models very hard to debug. The example model in Listing 1 performs $1 + 2 + \dots + N$ substitutions and simplifications in order to deduce that $a[1] = a[2] = \dots = a[n]$.

Listing 1. Alias Model with Poor Scaling

```

model AliasClass_N
  constant Integer N=60;
  Real a[N];
equation
  der(a[1]) = 1.0;
  a[2] = a[1];
  for i in 3:N loop
    a[i] = i*a[i-1]-sum(a[j]
      for j in 1:i-1);
  end for;
end AliasClass_N;

```

Using a real-world example, the Engine1a model from the Modelica MultiBody library, [12], the majority of equations have less than 10 operations (Figure 3), which is a manageable number to go through if one needs to debug a model and to find out which equations are problematic.

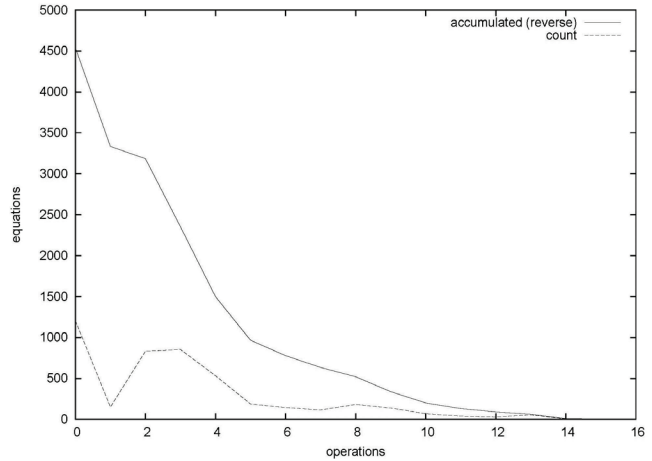


Figure 3. The number of symbolic operations performed on equations in the Engine1a model.

5.5 Runtime supported by static information

In order to produce better error messages during runtime, it is beneficial to be able to trace the source of the problem. The toy example in Listing 2 is used to show the information that the augmented runtime can display when an error occurs. The user should be presented with an error message from the solver (linear, nonlinear, ODE or algebraic does not matter). Here, the displayed error comes from the algebraic part of the solver. It clearly shows that $\log(0.0)$ is not defined and the source of the error in the concrete syntax (the Modelica code that the user may influence) as well as the name of the component (which may be used as a link by a graphical editor to quickly switch view to the diagram view of this component). The symbolic transformations performed on the equation are also displayed, which can help debug the model better.

Listing 2. Runtime Error

```
Error: At t=0.5, block1.u = 0.0 is not in
the domain of log (>0)
Source equation: [Math.mo :2490:9-2490:33]
y = log(u)
Source component: block1 (MyModel
Modelica.Blocks.Math.Log)
Flattened equation: block1.y = log(
block1.u)
Manipulated equation: y = log(u)
<Operations>
variable substitution: log(block1.u ) =
log(u)
<Depending on equations (from BLT)>
u <:link>
```

Currently we are working on extending the information we collect during the static analysis to build the Interactive Dependency Graph from Figure 1, Section 4.

6 Dynamic Debugging

6.1 Using the Algorithmic Code Debugger

The debugger part for algorithmic Modelica code is implemented within the OpenModelica environment as a debug plugin for the Modelica Development Tooling (MDT) which is a Modelica programming perspective for Eclipse. The Eclipse-based user interface of the new efficient debugger is depicted in Figure 4.

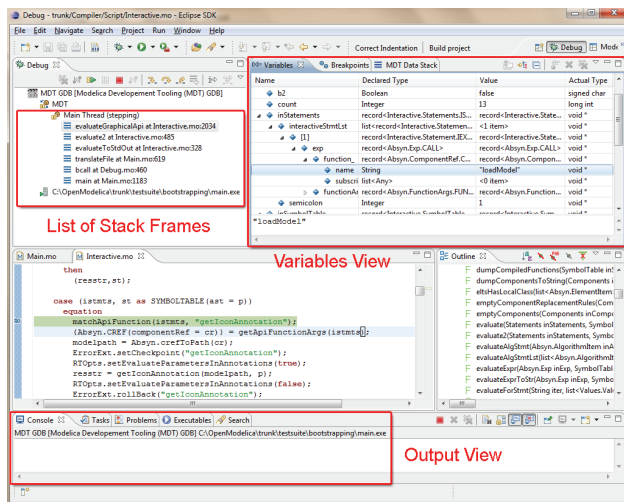


Figure 4. The debug view of the new efficient algorithmic code debugger within the MDT Eclipse plugin.

The algorithmic code debugger provides the following general functionalities:

- Adding/Removing breakpoints.
- Step Over – moves to the next line, skipping the function calls.
- Step In – steps into the called function.
- Step Return – completes the execution of the function and comes back to the point from where the function is called.

- Suspend – interrupts the running program.
- Resume – continues the execution from the most recent breakpoint.
- Terminate – stops the debugging session.

It is much faster and provides several stepping options compared to the old dynamic debugger because the old debugger was based on high-level source code instrumentation which made the code grow by a factor of the number of variables. The debug view primarily consists of two main views:

- Stack Frames View
- Variables View

The stack frame view, shown in Figure 5, shows a list of frames that indicates how the flow had moved from one function to another or from one file to another. This allows backtracing of the code.

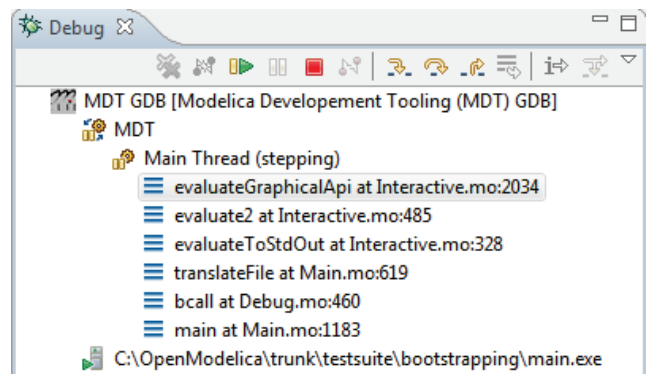


Figure 5. The stack frame view of the debugger.

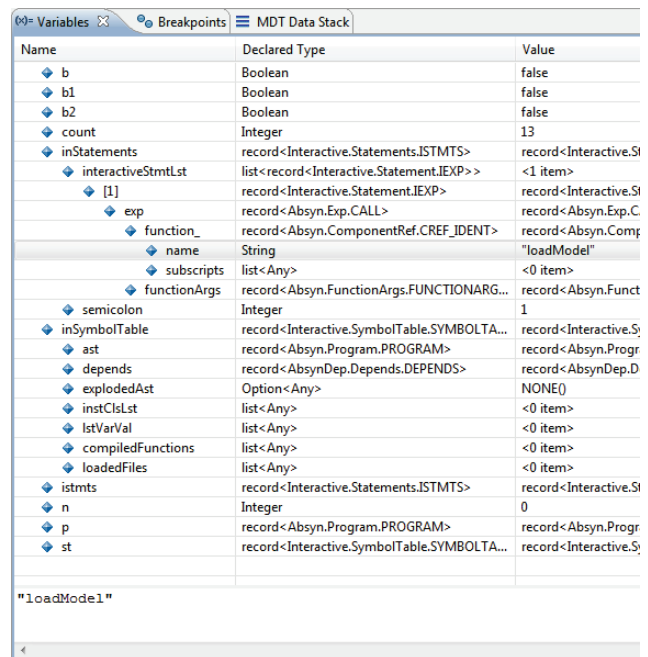


Figure 6. The variable view of the new debugger.

It is possible to select the previous frame in the stack and inspect the values of the variables in that frame.

However, it is not allowed to select any of the previous frames and start debugging from there.

Each frame is shown as `<function_name at file_name:line_number>`.

The Variables view (Figure 6) shows the list of variables at a certain point in the program. It contains four columns:

- Name – the variable name.
- Declared Type – the Modelica type of the variable.
- Value – the variable value.
- Actual Type – the mapped C type.

By preserving the stack frames and the variables it is possible to keep track of the variables values. If the value of any variable is changed while stepping then that variable will be highlighted yellow (the standard Eclipse way of showing the change).

6.2 Dynamic Debugger Implementation

In order to keep track of Modelica source code positions, the Modelica source-code line numbers are inserted into the transformed C source-code. This information is used by the Gnu Compiler GCC to create the debugging symbols that can be read by the Gnu debugger GDB [10].

Through the bootstrapped OpenModelica Compiler [4] the line number information is propagated all the way from the high level Modelica representation to the low level intermediate representation and the generated code.

This approach was developed for the symbolic model transformation debugger described in [5] and is also used in this debugger.

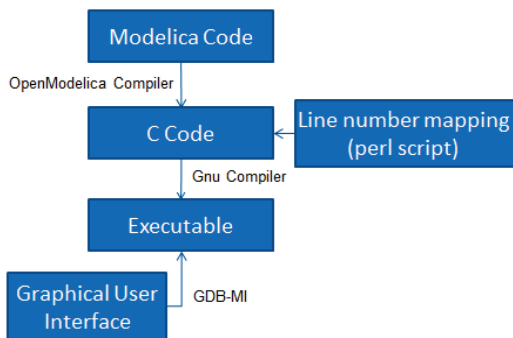


Figure 7. Dynamic debugger flow of control.

Consider the Modelica code shown in Figure 8:

```

HelloWorld.mo
function HelloWorld
  input Real x;
  output Real y;
algorithm
  y := sin(x);
end HelloWorld;
  
```

Figure 8. Modelica Code.

The OpenModelica Compiler compiles this HelloWorld function into the C source-code depicted in Figure 9.

```

HelloWorld.c
modelica_real _y;
modelica_real tmp2;
_tailrecursive:
/* functionBodyRegularFunction: out inits */
/* varOutput varInits(_y) */

/* functionBodyRegularFunction: state in */

/* functionBodyRegularFunction: var inits */
/* functionBodyRegularFunction: body */
/*#modelicaLine [HelloWorld.mo:5:3-5:14]*/
tmp2 = sin(_x);
_y = tmp2;
/*#endModelicaLine*/

_return:
/* functionBodyRegularFunction: out var copy */
/* functionBodyRegularFunction: state out */
/* functionBodyRegularFunction: out var assign */
/* varOutput varAssign(_y) */
tmp1.targ1 = _y;

/* GC: pop the mark! */
mmc_GC_undo_roots_state(mmc_GC_local_state);

/* functionBodyRegularFunction: return the outs */
return tmp1;
  
```

Figure 9. Generated C source-code.

The generated code contains blocks which represent the Modelica code lines. The blocks are mentioned as comments in the following format `/*#modelicaLine [modelica_source_file:line_number_info]*/`.

This information is now used to generate debug symbols that are recognized by GDB. The generated C source-code is used by a small Perl script to create another version of the same source-code with different line number blocks, see Figure 10.

```

HelloWorld.conv.c
#line 25 "HelloWorld.c"
/* functionBodyRegularFunction: state in */
#line 26 "HelloWorld.c"

#line 27 "HelloWorld.c"
/* functionBodyRegularFunction: var inits */
#line 28 "HelloWorld.c"
/* functionBodyRegularFunction: body */
#line 5 "/c/Users/adeas31/workspaceMDT/HelloWorld/HelloWorld.mo"
tmp2 = sin(_x);
#line 5 "/c/Users/adeas31/workspaceMDT/HelloWorld/HelloWorld.mo"
_y = tmp2;
#line 33 "HelloWorld.c"

#line 34 "HelloWorld.c"
_return:
#line 35 "HelloWorld.c"
/* functionBodyRegularFunction: out var copy */
#line 36 "HelloWorld.c"
/* functionBodyRegularFunction: state out */
  
```

Figure 10. Converted C source-code.

The converted C source-code contains a line number mapping between the generated C source-code and the actual Modelica source-code in the GDB specific format. Examine the lines starting with `#line` in Figure 10.

The executable is created from the converted C source-code and is debugged from the Eclipse-based

Modelica debugger which converts Modelica-related commands to low-level GDB commands at the C code level.

The Eclipse interface allows adding/removing breakpoints. The breakpoints are created by sending the `<-break-insert filename:linenumber>` command to GDB. At the moment only line number based breakpoints are supported. Other alternatives to set the breakpoints are; `<-break-insert function>`, `<-break-insert filename:function>`.

These program execution commands are asynchronous because they do not send back any acknowledgement. However, GDB raises signals;

- as a response to those asynchronous commands.
- for notifying program state.

The debugger uses the following signals to perform specific actions:

- *breakpoint-hit* – raised when a breakpoint is reached.
- *end-stepping-range* – raised when a step into or step over operations are finished.
- *function-finished* – raised when a step return operation is finished.

These signals are utilized by the debugger to extract the line number information and highlight the line in the source-code editor. They are also used as notifications for the debugger to start the routines to fetch the new values of the variables.

The suspend functionality which interrupts the running program is implemented in the following way. On Windows GDB interrupts do not work. Therefore a small program *BreakProcess* is written to allow interrupts on Windows. The debugger calls *BreakProcess* by passing it the process ID of the debugged program. *BreakProcess* then sends the *SIGTRAP* signal to the debugged program so that it will be interrupted. Interrupts on Linux and MAC are working by default.

The algorithmic code debugger is operational and works without performance degradation on large algorithmic Modelica/MetaModelica applications such as the OpenModelica compiler, with more than 100 000 lines of code.

The algorithmic code debugging framework graphical user interface is developed in Eclipse as a plugin that is integrated into the existing OpenModelica Modelica Development Tooling (MDT). The tracking of line number information and the runtime part of the debugging framework is implemented as part of the OpenModelica compiler and its simulation runtime.

The algorithmic code debugger currently supports the standard Modelica data types including arrays and records as well as all the additional MetaModelica data

types such as ragged arrays, lists, and tree data types. It supports algorithmic code debugging of both simulation code and MetaModelica code.

Furthermore, in order to make the debugging practical (as a function could be evaluated in a time step several hundred times) the debugger supports conditional breakpoints based on the time variable and/or hit count.

The algorithmic code debugger can be invoked from the model evaluation browser and it breaks at the execution of the selected function to allow the user to debug its execution.

7 Conclusions and Future Work

We have presented static and dynamic debugging methods to bridge the gap between the high abstraction level of equation-based object-oriented models compared to generated executable code. Moreover, an overview of typical sources of errors and possibilities for automatic error handling in the solver hierarchy has been presented.

Regarding static transformational debugging, a prototype design and implementation for tracing symbolic transformations and operations has been made in the OpenModelica Compiler. It is very efficient with an overhead of the order of 0.01%.

Regarding dynamic algorithmic code debugging, this part of the debugger is in operation and is being regularly used to debug very large applications such as the OpenModelica compiler with more than 100 000 lines of code. The user experience is very positive. It has been possible to quickly find bugs which previously were very difficult and time consuming to locate. The debugger is very quick and efficient even on very large applications, without noticeable delays compared to normal execution.

A design for an integrated static-dynamic debugging has been presented, where the dependency and origin information computed by the transformational debugger is used to map low-level executable code positions back to the original equations. Realizing the integrated design is work-in-progress and not yet completed.

To our knowledge, this is the first debugger for Modelica that has both static transformational symbolic debugging and dynamic algorithmic debugging.

8 Acknowledgements

This work has been supported by the Swedish Strategic Research Foundation in the EDOP and HIPo projects and Vinnova in the RTSIM and ITEA2 OPENPROD projects. The Open Source Modelica Consortium supports the OpenModelica work.

References

- [1] Adrian Pop and Peter Fritzson (2005). A Portable Debugger for Algorithmic Modelica Code. In Proceedings of the 4th International Modelica Conference, Hamburg, Germany.
- [2] Adrian Pop, Peter Fritzson, Andreas Remar, Elmir Jagudin, and David Akhvlediani (2006). OpenModelica Development Environment with Eclipse Integration for Browsing, Modeling, and Debugging. In Proc of the Modelica'2006, Vienna, Austria.
- [3] Adrian Pop, David Akhvlediani, and Peter Fritzson (2007). Towards Run-time Debugging of Equation-based Object-oriented Languages. In Proceedings of the 48th Scandinavian Conference on Simulation and Modeling (SIMS'2007), see <http://www.scan-sims.org>, <http://www.ep.liu.se>. Göteborg, Sweden.
- [4] Martin Sjölund, Peter Fritzson, and Adrian Pop (2011a). Bootstrapping a Modelica Compiler aiming at Modelica 4. In Proceedings of the 8th International Modelica Conference (Modelica'2011), Dresden, Germany.
- [5] Martin Sjölund and Peter Fritzson (2011b). Debugging Symbolic Transformations in Equation Systems. In Proceedings of the 4th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools, (EOOLT'2011), Zürich, Switzerland.
- [6] Peter Bunus and Peter Fritzson (2003). Semi-Automatic Fault Localization and Behavior Verification for Physical System Simulation Models. In Proceedings of the 18th IEEE International Conference on Automated Software Engineering, Montreal, Canada.
- [7] Peter Bunus (2004). Debugging Techniques for Equation-Based Languages. PhD Thesis. Department of Computer and Information Science, Linköping University.
- [8] Peter Fritzson (2004). *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*, 940 pp., ISBN 0-471-471631, Wiley-IEEE Press.
- [9] Peter Fritzson, Peter Aronsson, Håkan Lundvall, Kaj Nyström, Adrian Pop, Levon Saldamli, and David Broman (2005). The OpenModelica Modeling, Simulation, and Software Development Environment. In Simulation News Europe, 44/45.
- [10] Richard Stallman, Roland Pesch, Stan Shebs, et al. (2011). Debugging with GDB. Free Software Foundation. [online] Available at: <<http://unix.lsa.umich.edu/HPC201/refs/gdb.pdf>> [Accessed 30 October 2011].
- [11] Open Source Modelica Consortium. *OpenModelica System Documentation Version 1.8.1*, April 2012. <http://www.openmodelica.org>
- [12] Modelica Association. *The Modelica Language Specification Version 3.2*, March 24th 2010. <http://www.modelica.org>. Modelica Association. *Modelica Standard Library 3.1*. Aug. 2009. <http://www.modelica.org>.
- [13] Uri Ascher and Linda Petzold. Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations. Society for Industrial and Applied Mathematics, 1998.
- [14] Willi Braun, Lennart Ochel, and Bernhard Bachmann. Symbolically derived Jacobians using automatic differentiation - enhancement of the OpenModelica compiler. In Modelica'2011.
- [15] Sven Erik Mattsson and Gustaf Söderlind. Index reduction in differential algebraic equations using dummy derivatives. Siam Journal on Scientific Computing, 14:677--692, May 1993.
- [16] Conal Elliott. Beautiful differentiation. In International Conference on Functional Programming (ICFP), 2009.
- [17] Jens Frenkel, Christian Schubert, Günter Kunze, Peter Fritzson, and Adrian Pop. Towards a benchmark suite for Modelica compilers: Large models. In Modelica' 2011.
- [18] Roberto Parrotto, Johan Åkesson, and Francesco Casella. An XML representation of DAE systems obtained from continuous-time Modelica models. In Proceedings of the 3rd International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools, pages 91--98. Linköping University Electronic Press, October 2010.

A Modelica Sub- and Superset for Safety-Relevant Control Applications

Bernhard Thiele*
Bernhard.Thiele@dlr.de

Stefan-Alexander Schneider†
stefan-alexander.schneider@bmw.de

Pierre R. Mai‡
pmai@pmsf.de

* German Aerospace Center (DLR), Institute for Robotics and Mechatronics, Germany

† BMW AG, 80788 München, Germany

‡ PMSF IT Consulting, Marzling, Germany

Abstract

Fueled by the continuous, rapid progress within microelectronics, ever more intelligent and intricate functions are realized in mechatronic systems. To control the complexity associated with such designs, model-based control design methods are increasingly adapted in industry. Despite Modelica's obvious suitability to efficiently create appropriate high fidelity system models, the utilization of Modelica for developing discrete control functions is not yet wide spread. Adoption of Modelica for this task offers the potential for a seamless development methodology from the logical virtual model down to the technical system architecture, with corresponding traceability and maintainability benefits.

This contribution will specifically address this potential and propose a Modelica sub- and superset adequate for use within the development of safety-relevant control applications.

Keywords: embedded systems; functional safety; simulation; code generation; compiler; formal methods; validation; verification

1 Introduction

Model-based design has emerged as a standard development approach for the design of embedded systems. Its original promise to provide a more rapid and economic development process is confirmed in industrial practice [5].

More and more embedded software components are specified in models representing the so-called high-level application that is then automatically transformed (usually via embedded C-code) into binary code that is executable on the embedded target: Fig-

ure 1 shows a typical model-based development environment where the specification model is first designed using a next generation high-level, domain-oriented modeling tool. These specification models are typically enriched with implementation details and converted to so-called code generation models. A code generator converts the code generation model into C-code that a cross compiler translates to object code. The different object codes, including legacy and basic software code are then finally linked to a binary to be executed by an embedded target. This approach reduces the implementation effort and time, especially in iterative development workflows. Model-based development methods have a significant impact on the development process and the development environment with its tools.

With the increase of applications, and along with that of software size and complexity, model-based approaches have found their way into safety-relevant applications, especially in the aerospace and automotive domains. This evolution has thrust the safety impact of model-based development, especially with regard to high-level modeling and code generators, into the spotlight.

As described above, for practical purposes the process of the generation of the executable program from the model is mainly based on two development tools: the code generator and the cross compiler (including the cross linker). From an abstract point of view, this concatenation of these two compilers is again a (system) compiler, and can be treated by the same theory as a compiler that would translate directly from the code generation model to the executable code, see Figure 2. In the following all such translation tools will be denoted abstractly as *development tools*.

The generated C-code can be seen as intermediate representation of the model, because it is both output

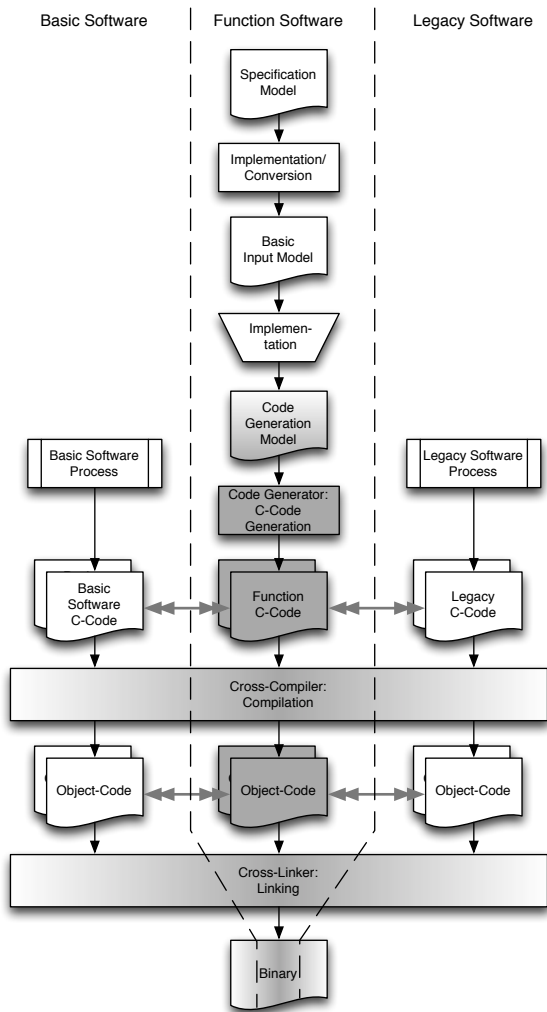


Figure 1: The generic build process for a model-based development toolchain with an automatic code generator (from [14]). The shaded parts indicate the tools and artifacts affected by what is later referred to as the *development tool* (code generator and cross compiler).

of the code generator as well as input to the cross compiler for the target. This perspective of the development tool is of central significance, because there is no need to perform any qualification activities on such internal representations as long as the C-code is only used as input to the cross compiler and is not further manipulated or used in any other activities that need to rely on the readability of the C-code. As a consequence no C-code reviews are needed in this case. This approach opens up the possibility to perform reviews on the model level. A main topic of this contribution is addressing the conditions that need to be established to allow such model level reviews.

Development tools may inject systematic faults into the executable program. Increasing the functional safety in this context means to minimize erroneous

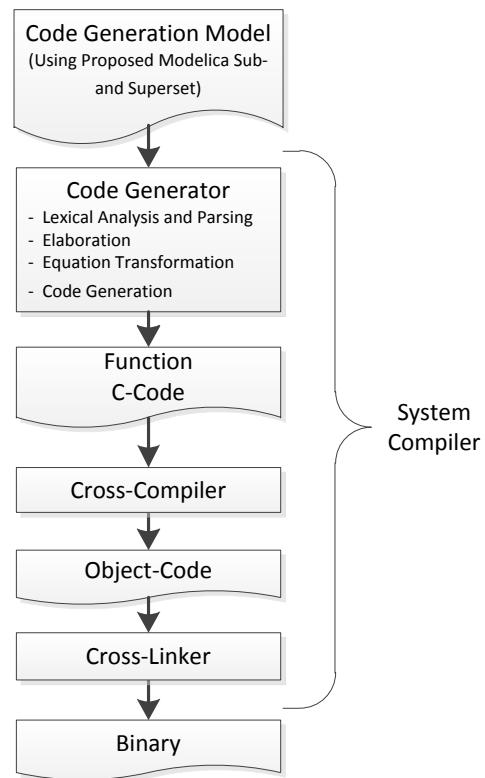


Figure 2: The concatenation of a code generator and a (cross) compiler can be treated as a (system) compiler that directly transforms from the code generation model to the executable (target) code. The Figure indicates that the system compiler is based on the proposed Modelica sub- and superset. Note that this system compilation process may be realized significantly different than a compilation process used in order to create *simulation* executables of models (see Section 4.2.5).

outputs of the development tools due to malfunctions and/or reliably detecting those erroneous outputs when they occur.

One method to gain confidence in a software development tool, is the validation of the software tool by a validation suite, which comprises a test suite specifically designed to exercise the development tool in ways that would provoke any systematic malfunctions.

In order to design a suitable modeling language, powerful development tools as well as an efficient and effective validation suite, it is important to understand precisely the role of translators and what the validation suite is intended to demonstrate. We therefore introduce some definitions in the context of a validation suite in Section 5, together with a discussion of the role of language structure and complexity.

For both the validation of the input language and

the transformation process, we have to cope with the curse of complexity. It is therefore of crucial importance to keep the language of the code generator models as simple and well-defined as possible, especially with regards to the number and complexity of basic constructs in the language, while also minimizing the number and complexity of performed transformation rules in the code generation process. This is especially true of transformation rules stemming from optimization rules. On the other hand the language so defined still has to be suitable for human consumption, so that the complexities of the code generation process are not just offloaded to the programmer.

Up to now the use of Modelica in embedded systems development is usually restricted as a modeling language for the physical plant dynamics. This can be attributed to:

1. A somewhat too limited expressiveness in modeling discrete controller functions.
2. The lack of a flexible, seamless development approach from the controller model comprising the *logical functions* to the *technical system architecture* (i.e., code running on the target platform).
3. And last but not least because safety-relevant software functions need means to achieve a high assurance level, which is not supported with current Modelica.

When discussing the use of Modelica in the context of control application, often advanced control concepts based on inverted plant dynamics are described [18], [19], [3]. Some Modelica tools are capable of automatically synthesising such controllers and generate code for them. This usually requires fairly sophisticated symbolic manipulation capabilities by the tool. For example, it may require to differentiate a subset of the equations, select appropriate states and solve the resulting system of differential and algebraic equations numerically.

However, the intrinsic complexity in the resulting control algorithms imposes an additional burden if the topmost design goal is in providing high assurance control systems. For the following discussion it is therefore assumed that the design trade-off is biased to prefer high assurance control over high performance control systems.

The aim of the paper is to study impacts of a safety-relevant development process (relying on validated tools) to high-level, domain-oriented modeling languages. In particular it proposes a sub- and superset

of the modeling language Modelica suitable for such safety-relevant software development activities. To illustrate the development using the proposed language elements a showcase library (referred to as SAFEDIS-CRETECONTROL library) is presented and applied at an exemplary use case.

2 Development Roles

Model-based development is an established method in the development of safety-relevant products. As seen above, especially the two transformations code generation and cross compilation play an important role. To ensure that the benefits of this approach have full effect the working mode of the development tools needs to be well understood.

The intended software development process, and hence the development environment, has to provide a balance between controlled process steps and flexibility of user access: On the one hand, the user may not be restricted too much and must still have principal control over all development activities – too many restrictions reduce the acceptance and thus also the productivity and quality of the work. On the other hand, too few restrictions lead to error-prone development practices, and ultimately to preventable faults in the software.

Various stake-holders participate in the development in different roles with different requirements and expectations. A suitable Modelica sub- and superset will have to support at least the following roles in the development process:

Role 1 - Developer of the Embedded Control System. This role requires a sufficiently expressive modeling language with sound language elements with clear semantics to design and test the intended functionality.

Role 2 - Tool Developer. This role requires the precise definition of the input modeling language: There should be no unclear corner cases in the semantics. The language should be efficiently compilable to target code.

Role 3 - Reviewer for Functional Safety. This role requires a clear and unambiguous description of the functionality, including all semantically relevant modeling details in compact form for efficient reviews. It should be possible to determine coverage at the model level, and allow for tracing of requirements to the relevant model parts.

Role 4 - Tool Qualifier. This role requires a suffi-

ciently small number of modeling elements with clear semantics as well as clear, ideally highly localized composition rules, in order to establish a validation suite for the development tool. The boundaries of the development tools, i.e., input and output notations, have to be clearly defined. Automated processes should ideally be separately testable, to minimize complexity. For more details again see Section 5.

These different roles have partly coincident — semantic aspects — and partly contradictory — expressiveness of the language — requirements to the development process and with that to the development tools and their modeling languages.

3 Requirements for a Safety Oriented Modelica Sub- and Superset

In this section we study the requirements of the above introduced roles to modeling language, to the development environment and their tools, and a validation suite for safety-relevant developments.

We will see later in this contribution that these requirements can be met only by specific restrictions and extensions of the modeling domain language. This finally leads us to the introduction of a sub- and superset of Modelica based on these requirements, that is simple in order to facilitate high assurance designs, yet expressive enough to allow modeling of many control strategies of practical relevance.

The textual representation of the Modelica language defines the full semantics of a given model, therefore we will begin with the requirements to the textual representation.

3.1 Requirements to the Textual Representation

At a first glance, we argue that functional reviews should be done at the textual Modelica language level, since the language semantics are specified at the textual level and graphical representation may hide important details. However, of course the graphical level provides an abstraction that eases comprehension of the intended model semantics and is hence an extremely valuable supplementary to the textual review. We therefore describe in the following section additional requirements to the graphical representation designed to avoid any hidden important details in the graphical representation. It is then left to the reviewer and his preferences either to perform a textual or a graphical review.

We start with a coincident requirement for the roles:

Requirement 1 - Formally Sound Language Set. The language sub- and superset must be formally sound, so that validation and verification methods e.g. formal methods can be supported. Required by Roles 1, 2, 3, and 4

Requirement 2 - Minimum Expressiveness of the Language Set. The language sub- and superset must have enough expressiveness to allow the clear and concise specification of discrete open- and closed-loop control algorithms and their related support logic. Required by Roles 1 and 2.

Requirement 3 - Target Data Types and Operations. The language should provide a mechanism that allows to extend its data types and operations to support fundamental data types and operations available on the embedded target platform. Required by Roles 1 and 2.

Requirement 4 - Target Code Generation. The language should permit automatic generation of target platform C-code¹ that is: a) efficient, b) avoids unsafe constructs², c) is traceable³, and d) integrates smoothly into embedded systems software architectures. Required by Roles 1, 2, and 1.

Requirement 5 - No Continuous-Time Dependencies. The language must not have dependencies to continuous-time system solver functionalities running in the background. Required mainly by Roles 1 and 2.

Requirement 6 - Compile Time Analysis. The language should allow compile time analysis of important properties in order to reject dubious programs (missing initial values, type checking, clock analysis, detection of cyclic definition that result in algebraic loops, etc.). Required mainly by Roles 1 and 3.

Requirement 7 - Modular Code Generation. The language must support modular code generation, i.e., within a model composed by connecting several blocks it must be possible to generate a transition function for each block definition and by composing them together produce the overall transition function. Required mainly by Roles 1 and 3.

¹Automatic C-code generation is stipulated, since C-code is the most popular language for targeting embedded systems and (certifiable) compilers are available. However direct-to-binary code generators are not precluded by this, and similar though not identical concerns arise for those cases.

²For example by conforming to coding standards like MISRA AC AGC [17].

³Given a fragment of the automatically generated C-code it must be possible to trace it back to the model elements that caused its generation.

Requirement 8 - Modular Initialization. As a consequence of Requirement 7 also initialization of (state) variables must be supported in a modular manner, i.e., initial values of (state) variables in modular blocks deduced during compile time analysis must not depend on the environment enclosing the block. Furthermore, if initial values can not be *uniquely* determined from the given constraints and set start values code generation shall abort with an error message. Required mainly by Roles 1, 3 and 4.

Requirement 9 - Tangible Fixation of Automatically Deduced Properties. In order to ensure reproducibility of code generation and reviewability⁴, it must be possible to fixate all properties of a model that influence code generation in a tangible, reviewable form. In particular it must be possible to fixate initial values that are automatically deduced by a tool, so that code generation will always use the fixated values instead of recalculating those values on the fly at the time of code generation. Required mainly by Role 3 and 4.

Requirement 10 - Manual Block Scheduling. Manual scheduling of block execution (as opposed to scheduling based on automatic causality analysis) must be possible on an optional basis. Required mainly by Role 1.

The following requirement is mainly motivated by the role of a tool qualifier and typically holds the most potential for discussion with the other roles, especially with the role developer:

Requirement 11 - Restricted Language Scope. To ease tool validation the language should be as simple and clear as possible. This shall be achieved by restricting the scope of the Modelica language to a (preferably small) *sub- and superset* relevant for the addressed problem domain, i.e. suitable for the implementation of the blocks in the SAFEDISCRETECONTROL library. Particularly, simplicity and clarity of the language sub- and superset is to be preferred over feature richness. Required mainly by Role 4.

As already mentioned above, in the following section we describe additional requirements to an optional graphical representation in order to perform a fully equivalent review on the graphical representation.

⁴Note that this requirement also enables separate validation of code generator and property-deduction code, since the fully fixated model provides the checkable interface between both processes.

3.2 Additional Requirements to the Graphical Representation

In computer science, semantics of a textual or graphical language refers to the meaning of programs written in it. Although the semantics of Modelica are described on a textual language level, Modelica provides standardized annotations for the graphical representation of models [10].

The main idea is that a *library* developer uses the textual Modelica language to code basic functionalities in components that are annotated with a graphical illustration, while an *application/model* developer (library user) works on a graphical level by just dragging, dropping and connecting the library components in order to compose the intended functionality.

How can we now avoid that not obvious or even hidden details in the graphical representation prevent the reviewers from performing an efficient and effective graphical review? In this section we therefore formulate *additional requirements to the graphical representation* that enable both developers (Role 1) and reviewers (Role 3) to *entirely work at a graphical level*.

Within Section 4.4 a conceptual library design (denoted SAFEDISCRETECONTROL library) is briefly presented that complies to the requirements stated in this section. In combination with adherence to the rules formulated in Section 4.3.2 the usage of such a library could then enable both, developers and reviewers, to entirely work at a graphical level.

We start with the graphical pendant to the textual requirement 1:

Requirement 12 - Intuitive Block Semantics. Blocks from SAFEDISCRETECONTROL and their compositions should not exhibit any behaviour which would be deemed surprising or non-obvious by a domain expert. Required by Roles 1, 2, 3, and 4.

The restriction on blocks reflects the textual requirement 11:

Requirement 13 - Restricted Set of Allowed Blocks. A high-level application model is only allowed to be composed from a set of thoroughly tested and validated basic blocks defined in the SAFEDISCRETECONTROL library. Required mainly by Roles 3 and 4.

Requirement 14 - Data Flow Semantic. Block diagrams with data flow semantic are used at the graphical level. Required by Roles 1, 2, 3, and 4.

Requirement 15 - Graphical Level Code Reviews. Code reviews of models should be feasible as far as possible at the graphical level. Consequently, any se-

manatics associated with the blocks from SAFEDISCRETECONTROL and their compositions should be completely evident by inspecting the graphical diagram layer, i.e., apart from clearly marked exceptional cases there should be no cases where the semantics of a model is not entirely and uniquely understandable from inspection of the block diagram. Required mainly by Role 3.

Requirement 16 - Block Testability. Any block within SAFEDISCRETECONTROL must be designed, so that extensive testing of the block is easily possible, i.e., simple, lean designs are preferred over sophisticated, complex designs. Note that this also implies that when balancing modeling comfort of blocks against simplicity, the bias is towards simplicity. Required mainly by Roles 3 and 4.

Requirement 17 - Composition Testability. Compositions of blocks from SAFEDISCRETECONTROL must again result in a block that is suitable for extensive testing, e.g., by restricting the number of inputs and outputs that are allowed for a block. Required mainly by Role 3 and 4.

Requirement 18 - Traceability. Compositions of blocks from SAFEDISCRETECONTROL must result in generated code that can be traced back to the blocks in the model, in order to easily perform, e.g., code coverage analysis on the target level but mirror back the results onto the model level. Required mainly by Role 3.

4 Proposal for a Safety Oriented Modelica Sub- and Superset

The aim of this section is to introduce a sub- and superset of Modelica that is simple in order to facilitate high assurance designs, yet expressive enough to allow modeling of many control strategies of practical relevance.

4.1 Terminology

The following list defines some key terms used subsequently.

Basic Blocks Blocks that have no inner instance of other blocks are subsequently referred to as *basic blocks*. These blocks may only contain parameters, connectors and (textual) equations.

Composite Blocks Blocks that are (graphically) composed from other blocks are subsequently re-

ferred to as *composite blocks*. These blocks may only be composed from other blocks connected by `connect(...)` equations. Therefore they do not contain any other textual equations.

Clocks Clocks provide an activation signal or *clock signal* used for synchronous scheduling of a set of equations activated by that clock signal. They recently entered the Modelica language standard.

Clock Blocks Special basic blocks containing clocks that provide a clock signal are subsequently referred to as *clock blocks*.

Atomic Blocks Blocks which are executed as a single unit (akin to a function call with input and output arguments) are referred to as *atomic blocks*.

4.2 Superset: Language Extension Proposal

To meet the requirements defined in Section 3 it is not sufficient to solely *restrict* the language elements to a *subset* of the current Modelica 3.3 standard specification. In addition it is necessary to *extend* the language elements, effectively forming a *superset* of the current language.

This section proposes several language extensions by

1. Explaining the perceived limitation of Modelica 3.3 that needs to be addressed.
2. Proposing a language extension that overcomes the limitation.

4.2.1 Data Types Extension

Modelica 3.3 [10, Section 12.9] specifies the following data type mapping to C:

Modelica data type	Default mapping to C
Real	double
Integer	int
Boolean	int
String	const char*
Enumeration	int

Embedded processors often need finer control about the used data type⁵. Again it is necessary to make a trade-off between feature completeness and validation costs. Validation effort will raise for every supported

⁵E.g., for increased memory efficiency or because the embedded system simply doesn't provide efficient support for that data type, e.g., an embedded system with a FPU (Floating Point Unit) that supports only single precision floating point arithmetic.

data type. In effect it needs to be checked whether the savings gained by supporting a particular data type (e.g., because a cheaper electronic control unit (ECU) can be used) outweighs the additional costs in (tool) validation.

The following section will propose a rather general mechanism to extend the standard Modelica data types with more low-level hardware encoding information. Note that although the SAFEDISCRETECONTROL library presented in Section 4.4 only supports a subset of the listed data types, the extension to additional data types is straight forward. However, the associated additional validation effort for any additional supported data type is considerable.

4.2.2 Proposal for Data Type Extension

The relevant part of the Modelica specification defining the basic data types is [10, Section 4.8]. The notation in the specification is adapted to extend the definition of the Real, Integer and Boolean data types⁶. The following predefined enumeration types are used for the definition.

```
type PlatformType = enumeration(
  UInt8 "8-bit unsigned integer",
  SInt8 "8-bit signed integer",
  UInt16 "16-bit unsigned integer",
  SInt16 "16-bit signed integer",
  UInt32 "32-bit unsigned integer",
  SInt32 "32-bit signed integer"
);
```

```
type PlatformRealType = enumeration(
  Float "IEEE 754 single precision
        floating type",
  Double "IEEE 754 double precision
         floating type",
);
```

Using the definitions above the predefined types of Modelica are extended with the additional attribute `platformType`. The rationale for not supporting an attribute is given in the corresponding footnote. Note that the types are defined with Modelica syntax although they are predefined, fundamental data types in Modelica.

```
type Real
  RealType value; /* Accessed
```

⁶In this work the dedicated support of fixed-point arithmetic is not (yet) considered. Note that if fixed-point arithmetic is required it is possible (though not convenient) to use the proposed Modelica language extensions to implement and validate custom basic blocks that provide the required functionality.

```
  without dot-notation */
parameter StringType quantity;6
parameter StringType unit;
parameter StringType displayUnit;6
parameter RealType min=-Inf, max=+Inf;
parameter RealType start = NaN;7
parameter BooleanType fixed;8
parameter RealType nominal;9
parameter StateSelect stateSelect;10
parameter PlatformRealType platformType
  = PlatformRealType.Double;
end Real;
```

```
type Integer
  IntegerType value; /* Accessed
  without dot-notation */
parameter StringType quantity;6
parameter IntegerType min=-Inf, max=+Inf;
parameter IntegerType start = +Inf;7
parameter BooleanType fixed;8
parameter PlatformType platformType
  = PlatformType.Sint32;
end Integer;
```

```
type Boolean
  BooleanType value; /* Accessed
  without dot-notation */
parameter StringType quantity;6
parameter BooleanType start = false;7
parameter BooleanType fixed;8
parameter PlatformType platformType
  = PlatformType.Sint32;
end Boolean;
```

Note that the values of the variables may not be directly manipulated in memory and consequently there are no access routines.

4.2.3 Activation of Discrete-time Equations in Modelica

Before the recently released Modelica 3.3 language standard the activation of discrete-time equations was either due to *time events* or *state events*.

⁶Omitted for the sake of language simplification (Requirement 11).

⁷If no start value is given, the start value is deduced (in compliance with Requirement 8) during compile time analysis.

⁸The Attribute "fixed" cannot be applied on clocked discrete-time variables. It is true for variables to which the previous() operator is applied, otherwise false [10, Section 16.9].

⁹Nominal values are only useful in the context of numerical solvers. They have no relevance in our targeted discrete applications.

¹⁰Only useful for solving (continuous) differential equation systems.

Time events are scheduled by the solver along a global simulation time line. Time is a (physical) real number (as opposed to the principle of *multi-form time*¹² adapted by synchronous languages) that steadily increases during execution (simulation) of a Modelica model. The global simulation time can be accessed anywhere in a Modelica model by the built-in variable `time`¹³.

State events are detected by the solver if a variable (controlled by the solver) experiences a zero-crossing.

The event handling approach of Modelica works well for *simulating* a plethora of hybrid system models, but it has shortcomings if embedded systems code shall be generated from a Modelica model. The prerequisite that an “omniscient” solver “running in the background” detects and schedules events in order to activate the evaluation of a set of equations impedes straightforward integration into external environments.

In order to allow smooth integration of code generated from Modelica into embedded systems software projects, Modelica needs to allow *external* code to simply cause the evaluation of a set of (discrete-time) Modelica equations (without the internal participation of a hybrid systems solver that tries to detect whether the equations shall be evaluated or not). Nikoukhah and Furic [11] provide a notable discussion about the missing feature of external activation in context of using Modelica models within the Scicos¹⁴ modeling environment which similarly applies to using Modelica models in embedded systems software projects.

To allow external activation of Modelica models Nikoukhah and Furic propose in [11] to add an Event type to the Modelica language and discuss the elements and semantics needed to integrate that new type in a general and backwards compatible way¹⁵.

The latest Modelica 3.3 language standard added *synchronous language elements* particularly targeted at the implementation of control systems [10, Chapter 16, *Synchronous Language Elements*]. They add *clock activation* as a third way of activating discrete-time

equations that largely solves the hitherto criticised deficiencies.

Another notable advantage of clock activation in comparison to activation through the traditional state and time events mechanism is the support of *clock inference*. It is no longer necessary to explicitly propagate an event to all (block) instances that contain equations that should be activated by that event. The property of a variable that is explicitly *associated with a clock* is propagated to other variables that are related with that variable through equation relations. The usage of variables associated with different clocks within the same expression requires special *clock conversion operators*, otherwise it is a model error. This increases the modeling comfort and protects against modeling errors related to unconscious combination of signals sampled at different points in time.

The following section will use a subset of the synchronous language elements as a base to realize a mechanism that, sloppily speaking, allows to call blocks as functions. On the one hand the proposal will restrict the allowed set of synchronous language elements to a subset (for language simplification reasons), on the other hand it will introduce a slight extension in order to satisfy two use cases:

1. Allow smooth integration of generated code into external environments, e.g., AUTOSAR authoring environments.
2. Allow manual scheduling of block execution as depicted in Figure 3.

The requirement to allow manual scheduling of block activation might appear strange, since a program can figure out the “correct” activation sequence easily from the data flow. However interaction with external software components, as well as execution time and real-time requirements, can place additional restrictions on the activation sequence that can not be determined by data flow only. Therefore, manual scheduling can be necessary. Additionally, if discussing a safety-relevant design with authorities it can often be beneficial to document that a human being has thought of the correct activation sequence rather than a machine.

4.2.4 Proposal for Atomic and Priority Based Activation

Conceptually, the *atomic block* definition in Section 4.1 yields the semantic depicted in Figure 4.

¹²The multi-form time principle states that any sequence of events can be considered as a time scale for the reactive system that perceives these events.

¹³Note that at the beginning of Section 4.2 it is stated that the built-in variable `time` is not supported for the SAFEDISCRETE-CONTROL library.

¹⁴Scicos is a graphical dynamical system modeler and simulator with support for continuous and discrete time models (<http://www.scicos.org/>).

¹⁵An early draft version of this document actually proposed an activation mechanism inspired by the proposal of Nikoukhah and Furic.

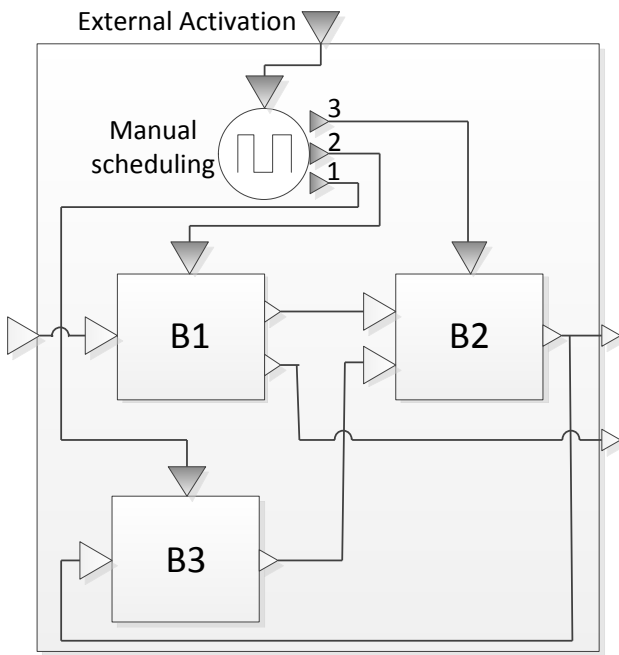


Figure 3: Manual scheduling of blocks.

Since an atomic block is executed as single unit it is required that all equations within the block must be activated from the same clock signal. However, it is possible that a block internally subsamples a clock signal and provides it as an output. Note that due to clock inference it is not necessary to provide an explicit clock input to every atomic block within a diagram as long as a unique clock can be inferred for it.

Atomic Blocks Currently there is no language support for treating a block as atomic according to our definition. To mitigate that deficiency the prefix **atomic** is proposed. The atomicity of a block is defined at the instance declaration.

atomic BlockModule a;

Akin to the execution of algorithms in Modelica models, an atomic block can be conceptually viewed as an atomic vector-equation (potentially with internal state) that maps its inputs to outputs, e.g.,

(out1,out2,...) = BlockModule(in1,in2,...);

Figure 5 further illustrates the difference between conventional and atomic block semantics.

Clock Priorities To allow manual scheduling of blocks it is proposed to extend the **subSample**(u, factor) operator with an (optional) additional integer argument denoting the priority of a clock:

subSample(u, factor, priority)

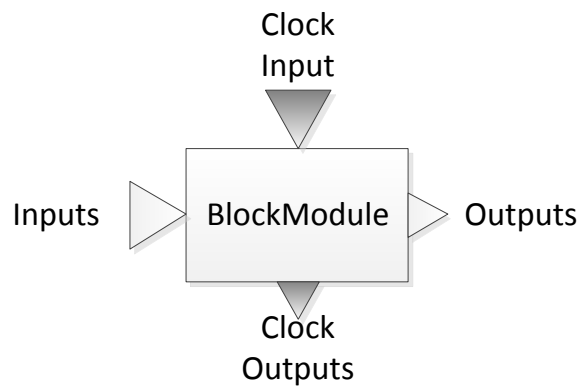


Figure 4: Conceptual atomic MIMO-block with data inputs and outputs, as well as one clock signal input (for activating the block) and (potential) several sub-sampled clock signal outputs.

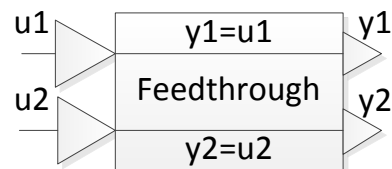


Figure 5: Trivial feedthrough block. *If declared atomic*, it is required that input signals u_1 and u_2 are active *at the same clock ticks* allowing to conceptually transform the block to a (periodically called) function $(y_1, y_2) = \text{Feedthrough}(u_1, u_2)$ (the block hierarchy is maintained at execution level). *If not declared atomic*, $u_1=y_1$ and $u_2=y_2$ are allowed to be *active at completely unrelated points in time* (the block hierarchy is flattened, see Section 4.2.5)!

Values assigned to **priority** must be positive (including zero), lower values indicate a higher priority. If omitted, the argument defaults to "priority = 0". The priorities are always *relative* to the source clock signal.

An example implementation for the scheduler block in Figure 3 is given below.

```

block Scheduler
input Clock clk;
output Clock clk1;
output Clock clk2;
output Clock clk3;
equation
  clk1 = subSample(clk, 1, 1);
  clk2 = subSample(clk, 1, 2);
  clk3 = subSample(clk, 1, 3);
end Scheduler;
  
```

The semantics is that the equations activated by a

higher priority clock must be executed first¹⁶. Note that due to the relative nature of clock priorities stated above, it is not possible that a clock has a higher absolute priority than the input clock. Furthermore, in order to uniquely associate every variable with one clock a `subSample(u)`¹⁷ operator needs to be present between the block connection shown in Figure 3. That has not been depicted to keep the diagram well-arranged.

Direct Block Activation The following language extension proposal is not essential to meet the requirements, however it supports more clearly arranged models.

The current language standard does not allow to directly assign a clock to a block with the semantics that all equations and variables in the scope of that block are marked to be part of the same base-clock partition. Therefore, the designated way to execute clocked equations within a controller block is by providing the clocking information at the inputs of that block and rely on clock inference. If the block needs several inputs that may result in a diagram like depicted in Figure 6.

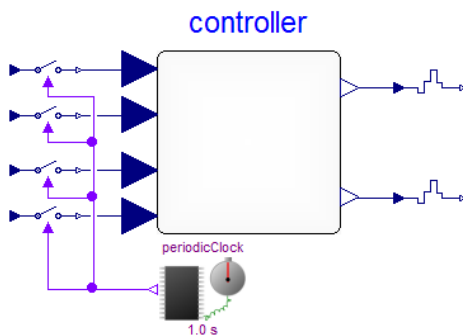


Figure 6: The designated way to execute clocked equations within a controller block using Modelica 3.3 is by providing the clocking information at the inputs of that block (by using the `sample(u, c)` operator and passing in the clock by the second argument) and rely on clock inference to forward the clocking information.

A tentative more explicit mechanism would be to introduce a built-in attribute “clock” for block classes that allows to assign a clock to a block. The semantics would be, that any variable and equation enclosed by

¹⁶The execution order of equations which are activated by clocks with equal priority is determined by the (standard) causality analysis algorithms of the Modelica tool.

¹⁷If the arguments “factor” and “priority” are not provided or zero, they are inferred.

the block would be associated with the assigned clock, e.g.,

```
block Controller
  input Real u;
  output Real y;
equation
  u = y;
end Controller;
```

```
model Environment
  Clock clk = Clock(0.5);
  Real s = sin(time);
  // associate c.u and c.y with clk
  Controller c(clock=clk);
equation
  c.u = sample(s);
end Environment;
```

However, in order to keep extensions to the official Modelica standard at a minimum this extension is not part of the proposed Modelica superset.

Clock Blocks and Interaction with the Physical Environment

Clock blocks provide clock signals. They are *source* blocks, since they need no input to provide a clock signal as output. In order to convert between continuous-time (physical environment) and clocked discrete-time signals the operators `sample(u)` (continuous-time variable `u` converted to discrete-time) and `hold(u)` (zero-order hold conversion of discrete-time variable `u` to continuous-time) are needed.

Note that according to Section 4.3, Rule 9 clock blocks as well as the conversion operators may not be part of the high-level application intended for code generation. They are elements needed to *simulate* the execution of the high-level applications within the simulation tool (depicted in Figure 7). Or formulated differently, they are idealized models of the environment that will execute the high-level application running on the ECU, e.g., a periodic scheduler of an operating system that activates the high-level application task.

Since clock blocks, `sample(u)` and `hold(u)` reside outside the high-level application they are not part of the Modelica sub- and superset proposed for code generation!

The Modelica Specification [10, Section 16.3] defines several overloaded `Clock(..)` constructors. A simple clock source block is modeled below.

```
block PeriodicClock
  parameter Real sampleRate = 0.1;
  output Clock y = Clock(sampleRate);
end PeriodicClock;
```

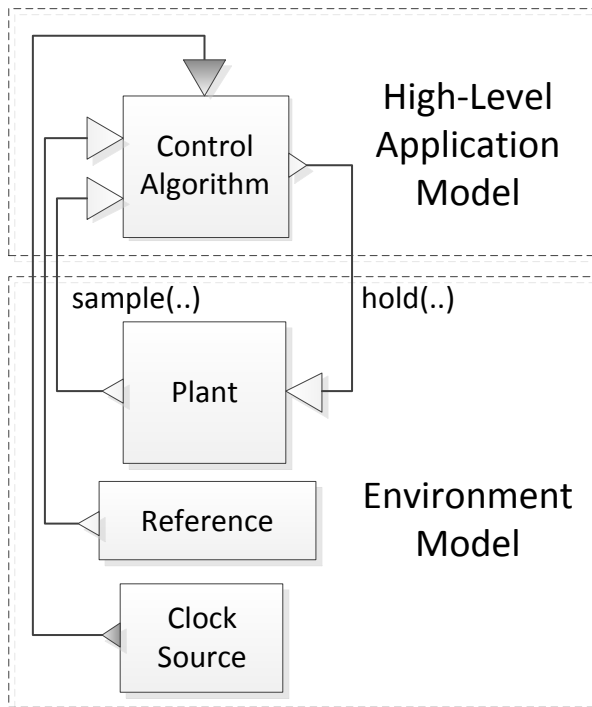



Figure 7: Simulation of a high-level application model using a clock block to model the execution of the application by its environment, e.g., by an operating system scheduler.

Calling a Block as a Function Combining the proposed priority based clock activation with modular code generation (Section 4.2.6) fulfils the requirement to “call blocks as functions” (stated in Section 4.2.3).

To exemplify, assume B1, B2 and B3 from Figure 3 have the annotation `Inline=false` and the manual scheduler is modeled by assigning appropriate priorities to the clock signals. A tool could generate following (conceptual) C-code.

```
double EnclosingBlock_B2_y = 0;

void EnclosingBlock(double u,
                    double* y1, double* y2) {
    double B1_y1, B1_y2;
    double B2_y, B3_y;
    B3(EnclosingBlock_B2_y, &B3_y);
    B1(u, &B1_y1, &B1_y2);
    B2(B1_y1, B3_y1,
        &EnclosingBlock_B2_y);
    *y1 = EnclosingBlock_B2_y;
    *y2 = B1_y2;
}
```

4.2.5 Typical Modelica Code Generation

The typical Modelica code generation process differs significantly from the automatic target code generation

intended for safety related applications. This section will give a short overview over the typical code generation process in order to better appreciate and understand the proposal for simplified and modular code generation presented in Section 4.2.6.

Compiling Modelica code usually involves substantial code transformation. Figure 8 gives an overview of the compilation and simulation process as described by Broman [4, p. 29].

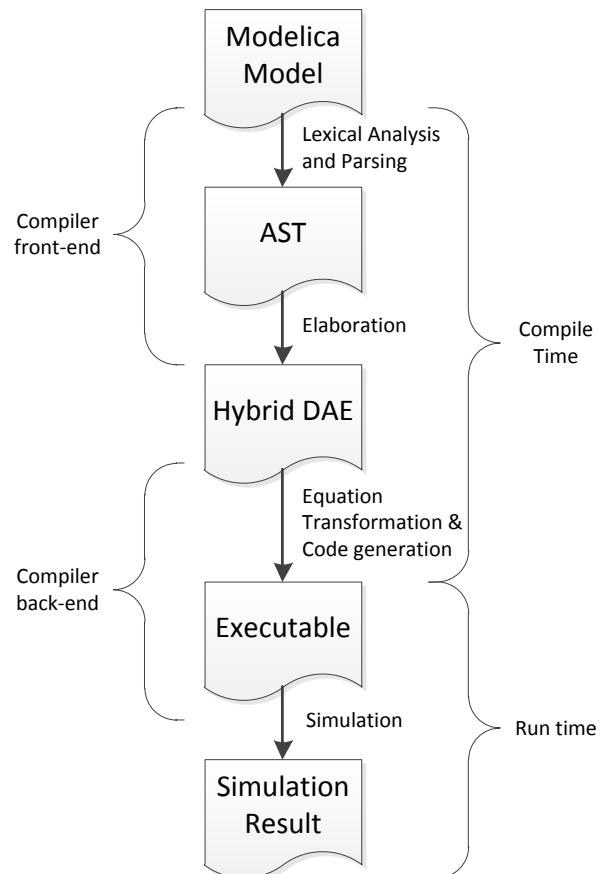


Figure 8: Outline of a typical compilation and simulation process for a Modelica language tool [4, p. 29].

The different phases are:

Lexical Analysis and Parsing This is standard compiler technology.

Elaboration Involves *type checking*, *collapsing the instance hierarchy* and *generation of connection equations* from connect-equations. The result is a hybrid DAE (consisting of variable declarations, equations from equations sections, algorithm sections, and *when*-clauses for triggering discrete-time behaviour).

Equation Transformation This step encompasses transforming and manipulating the equation sys-

tem into a representation that can be efficiently solved by a numerical solver. Depending on the intended solver the DAE is typically reduced to an index one problem (in case of a DAE solver) or to an ODE form (in case of numerical integration methods like Euler or Runge-Kutta).

Code generation For efficiency reasons tools typically allow (or require) translation of the residual function (for an DAE) or the right-hand side of an equation system (for an ODE) to C-code that is compiled and linked together with a numerical solver into an executable file.

Simulation Execution of the (compiled) model. While execution the simulation results are typically written into a file for later analysis.

In the context of code generation for safety relevant systems the typical processing of Modelica models has two problems:

1. In the **Elaboration phase** the instance hierarchy of the hierarchically composed model is collapsed and *flattened* into one (large) system of equations, which is subsequently translated into one (large) chunk of C-code inhibiting modularisation and traceability at the C-code level. That conflicts with Requirements 7, 8 and 18.
2. In the **Equation Transformation phase** the equations are extensively manipulated, optimized and transformed on the global model level. The algorithms used in this step are the core elements that differentiate the tools (*quality of implementation*). Although the basic algorithms are documented in the literature, the optimized algorithms and heuristics used in commercial implementations are a vendor secret. The lack of transparency and simplicity exacerbates tool qualification efforts.

Therefore, the compilation process for *simulation* may be significant different to the *target code compilation* process depicted in Figure 2. Not only because different compilers are used, but also because the target code generator may (need to) be an entirely distinct piece of software that may share only minimal to no amounts of code with the simulation code generator. In particular the target code generator depicted in Figure 2 is only required to understand the sub- and superset of the Modelica language intended for (discrete) software application models.

4.2.6 Proposal for Simplified and Modular Code Generation

In the context of safety related function development it is proposed to

1. Use simplified and transparent equation transformation algorithms for block diagrams.
2. Support modular code generation for blocks.

Simplified Transformation Algorithms The complexity in the compilation process of Modelica models is mainly due to *acausal*¹⁸, physical modeling. Transforming typical physical models in a form that can be efficiently solved by a numerical solver requires advanced symbolic manipulation techniques.

However, the proposed Modelica subset allows for a hugely simplified compilation. Recall, that compared to full Modelica, the following restrictions apply to the block diagram subset of Modelica proposed in this paper:

- Modelica block diagrams allow only *causal* connections between blocks.
- Only difference equations are permitted by the proposed language subset (the `der(. .)` operator is not available in the proposed language subset).

Transforming these equations to (causal) serial code is completely feasible by resorting to well known, published algorithms¹⁹ without needing additional expert knowledge to perform challenging tasks like index reduction (which is needed for almost all physical Modelica models of practical relevancy). Therefore, to increase the transparency of the transformation process it is proposed to use plain and open transformation algorithms suitable for the targeted Modelica subset.

Modular Code Generation Modular, or separate, code generation for blocks improves *traceability* within a code generation process, a key requirement when developing safety related functions [2, 1, p. 75].

The aim of modular code generation is to produce a transition function for each block definition and compose them together to produce the main transition function. However, flattening each block and manipulating the corresponding equation system on the global

¹⁸The term *acausal* in Modelica is somehow similar to what is referred in computer science as *descriptive*.

¹⁹For example by employing basic “Modelica” algorithms for causalization of an equation system into a block lower triangular form [12, 15, 6].

level usually allows to generate a better optimized, more efficient code. Consequently, a *trade-off* between *efficiency* and *traceability* is required [1, p. 75].

Instead of collapsing the instance hierarchy, as typically done within the **Elaboration phase**, it is proposed to provide an option that preserves the modularity of an instantiated block.

The Modelica specification already knows of annotations that can influence code generation [10, Section 18.3]:

```
code_annotation:
  annotation(" codeGenerationFlag "="
    { false | true } ")

codeGenerationFlag:
  "Evaluate" | "HideResult" | "Inline" |
  "LateInline" | "GenerateEvents"
```

Within the specification the effect of the flag "Inline" is limited to function declarations. We propose to extend that scope in order to use the flag to annotate block instances and block declarations that have been declared "atomic". An annotation at the block instance takes precedence over an annotation at the block declaration. If the flag is not explicitly set it defaults to **Inline=false**.

Therefore, for the example from Figure 9 the declaration to enforce separate code generation for the block instance writes:

```
atomic BlockModule blockModule
  annotation(Inline=false);
```

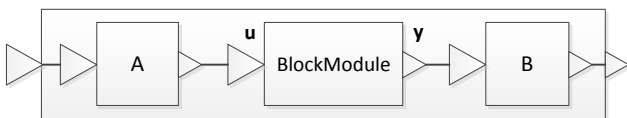


Figure 9: For every block it must be possible to *optionally* state whether the modularity of the block instance used in a composition is to be preserved at source code level.

For the BlockModule example from Figure 9 a C-function with suitable input and output data structures could be generated in way similar to

```
void BlockModule(inBlockModule_u *u,
                 outBlockModule_y *y);
```

The internal state variables of the block (if any) could be either part of the output data structure, or alternatively could be provided as a third argument to the function.

A common alternative approach is to use global variables with a suitable naming scheme to avoid

variable clashes for input, output, and state variables which results in functions with a void signature, e.g.,

```
void BlockModule(void);
```

If a suitable communication mechanism exists the code may also instead of directly accessing the variables use the communication interfaces provided by the run-time environment, e.g.,

```
void BlockModule(void) {
  inBlockModule_u u =
    get_inBlockModule_u();
  outBlockModule_y y;
  /* .. */
  set_outBlockModule_y(y);
}
```

Clarity may improve if for initialization or reset of state variables an additional, dedicated function is generated.

Additional Remarks on Code Generation A detailed discussion about automatic target code generation by Modelica tools is out of scope of this article. With no doubt the user needs to have more influence on the code generation than the options given by the proposal above. Customized control over some aspects of the code generation might be provided within the model in form of Modelica annotations (standardized or tool specific) or also at completely different locations and in different forms.

4.3 Subset: Reducing Language Complexity

The practice of defining *modeling* or *coding standards* for safety-relevant software projects is well established²⁰. This section proposes several rules for the development of safety-relevant control applications, akin to a modeling standard, in order to reduce the complexity of the language.

The rules *restrict* the allowed language elements, and hence define a *language subset*.

4.3.1 Textual Language Rules

To meet the requirements defined in Section 3 it is not sufficient to solely *restrict* the language elements to a *subset* of the current Modelica 3.3 standard specification. In addition it is necessary to *extend* the language

²⁰An example is the functional safety standard IEC 61508-3, in which the use of coding standards is highly recommended for SIL 3 and above [9]. Several coding/modeling guides published by The Motor Industry Software Reliability Association (MISRA) provide standards specifically targeted (but not limited) at the automotive industry, the most famous one being MISRA C [16].

elements, effectively forming a *superset* of the current language.

A language extension capable of meeting the requirements is proposed in Section 4.2. The following rule definitions require that this extension is available.

Rule 1 - Clocked Variables Exclusivity. Occurring variables and equations must be part of (discrete-time) clocked partitions. Note that this restriction implies that all allowed high-level applications have a purely time-discrete nature.

Rationale: Clocked variables and equations were introduced in Modelica 3.3 to provide improved support for implementation of (discrete-time) control systems.

Trace: Requirement 4, 5.

Rule 2 - Reduced Set of Keywords. Table 1 reproduces the keywords from the Modelica specification [10, Section 2.3.3]. Keywords that are not allowed in the proposed Modelica subset are stroked through. The semantics of the remaining elements are maintained appropriately²¹.

Rationale: Language simplification.

Trace: Requirement 11.

Rule 3 - Reduced and Restricted Set of Operators. The available Modelica operators are slightly reduced (no `.*` `./` `.*` `.-`) and the arithmetic operators are restricted to scalar types (see Table 2).

Rationale: Language simplification.

Trace: Requirement 11.

Rule 4 - Reduced Set of Built-in Functions and Operators with Function Syntax. Table 3 specifies the subset of supported built-in functions and operators with function syntax defined in [10, Section 3.7 and Chapter 10, 16 and 17].

Rationale: Language simplification.

Trace: Requirement 11.

Rule 5 - Supported Data Types. The scalar data types Boolean, Real and Integer are fully supported and extended to support more fine grain control about the underlying hardware encoding in Section 4.2.1. Enumeration is not supported, the support of String is limited to parameter values and constants. Array support is limited. Most (overloaded) operators and built-in functions related to arrays are not supported (see Table 2 and 3).

Rationale: Language simplification, as well as increase of language expressiveness to satisfy common

data type requirements from the embedded systems domain. Functionality provided by the array operators and built-in functions, e.g., scalar product, can be programmed by using the scalar operators and loops.

Trace: Requirement 4, 11.

Rule 6 - No Support of Built-in Variable time. The built-in variable `time` (see [10, Section 3.6.7]) is not supported.

Rationale: Physical time is a quantity of continuous system simulation and therefore not supported in the time-discrete language subset. If an absolute wall-clock time is needed in the application logic, it has to be passed in from the external environment as a (Real) input signal.

Trace: Requirement 5.

Table 1: Reduced set of allowed Modelica keywords.

algorithm	discrete	false	loop	record
and	each	final	model	pure
annotation	else	flow	not	redeclare
assert	elseif	for	operator	replaceable
block	elsewhen	function	or	return
break	encapsulated	if	outer	stream
class	end	import	output	then
connect	enumeration	impure	package	true
connector	equation	in	parameter	type
constant	expandable	initial	partial	when
constrainedby	extends	inner	protected	while
der	external	input	public	within

Table 2: Reduced set of allowed operators

Operator Group	Operator Syntax
<i>postfix array index operator:</i>	<code>[]</code>
<i>postfix access operator:</i>	<code>.</code>
<i>postfix function call:</i>	<code>funcName(. .)</code>
<i>exponentiation:</i>	<code>^</code>
<i>multiplicative^a:</i>	<code>*</code> <code>/</code> <code>.*</code> <code>./</code>
<i>additive^a:</i>	<code>+</code> <code>-</code> <code>+expr</code> <code>-expr</code> <code>++</code> <code>--</code>
<i>relational:</i>	<code><</code> <code><=</code> <code>></code> <code>>=</code> <code>==</code> <code><></code>
<i>unary negation:</i>	<code>not expr</code>
<i>logical and:</i>	<code>and</code>
<i>logical or:</i>	<code>or</code>
<i>array range:</i>	<code>expr : expr</code>
<i>conditional:</i>	<code>expr : expr : expr</code>
<i>named argument:</i>	<code>if expr then expr else expr</code> <code>ident = expr</code>

^a Note that contrary to [10, Section 3.4] the arithmetic operators `^` `*` `/` `+` `-` are limited to operate on scalar types only and the elementwise operators `.*` `./` `.*` `.-` are not available.

4.3.2 Additional Graphical Representation Rules

The following rules establish a modeling standard for the graphical representation of Modelica models, targeting the requirements formulated in Section 3.2, to

²¹Note that the reason for excluding a keyword is not because it would be unsafe to allow it. The reasons for excluding keywords is to reduce the complexity of the language as much as possible down to a set of (indispensable) core elements.

Table 3: Reduced set of built-in functions and built-in operators with function syntax

<i>Numeric Functions and Conversion Functions</i>	abs(v) sqrt(v) String(..)	sign(v) Integer(e)
<i>Event Triggering Mathematical Functions^a</i>	div(x,y) rem(x,y) floor(x)	mod(x,y) ceil(x) integer(x)
<i>Built-in Mathematical Functions and External Built-in Functions</i>	sin(x) tan(x) acos(x) atan2(x,y) cosh(x) exp(x) log10(x)	cos(x) asin(x) atan(x) sinh(x) tanh(x) log(x)
<i>Derivative and Special Purpose Operators with Function Syntax</i>	der(expr) cardinality(e) semiLinear(..) actualStream(v) getInstanceName()	delay(..) homotopy(..) inStream(v) spatialDistribution(..)
<i>Event-Related Operators with Function Syntax</i>	initial() noEvent(expr) sample(s,i) edge(b) reinit(x, expr)	terminal() smooth(p, expr) pre(v) change(v)
<i>Synchronous Language Elements</i>	Clock() previous(u) hold(u) ^b superSample(..) backSample(..) interval(u)	Clock(..) ^b sample(u, c) ^b subSample(..) shiftSample(..) noClock(u)
<i>State Machines^c</i>	transition(..) activeState(state) timeInState()	initialState(state) ticksInState()
<i>Array Dimension and Size Functions</i>	ndims(A) size(A)	size(A,i)
<i>Dimensionality Conversion Functions</i>	scalar(A) matrix(A)	vector(A)
<i>Specialized Array Constructor Functions</i>	identity(n) zeros(..) fill(..)	diagonal(v) ones(..) linspace(x1,x2,n)
<i>Reduction Functions and Operators</i>	min(..) sum(..)	max(..) product(..)
<i>Matrix and Vector Algebra Functions</i>	transpose(A) symmetric(A) skew(x)	outerProduct(v1,v2) cross(x,y)
<i>Array Constructor and Concatination</i>	array(..)	cat(..)

^a No events are triggered from these functions for the proposed language subset (see [10, Section 16.8.1]).

^b Only the “Inferred Clock” operator variant `Clock()` is supported. The other `Clock(..)` constructors as well as the `sample(u, c)` and `hold(u)` operators are not part of the language subset proposed for embedded target code generation (see Section 4.2.4 “Clock Blocks and Interaction with the Physical Environment”).

^c Present proposal excludes state machines (see Rule 7).

enable development and reviews to be conducted entirely on the graphical level.

Rule 7 - Block Diagrams Only. The present proposal focuses on the support of *block diagrams* and excludes *state diagrams*.

Rationale: This is to limit the required effort and associated complexity. Introduction of expressive state diagrams that integrate naturally with block diagrams and allow generation of efficient and safe code is a huge effort in its own.

Trace: Requirement 14.

Rule 8 - Causal Connectors Exclusivity. Only causal Connectors are allowed.

Rationale: Corollary to Rule 7

Rule 9 - Atomic Blocks for High-level Application.

The use of atomic blocks is suggested for the top-level hierarchies of a model that shall be automatically translated into embedded C-code.

Rationale: Enables a clean and clear execution model, as well as an obvious translation of a block to a function call (see Section 4.2.6).

Trace: Requirements 4, 7, 8, 12, 15, 16, 17, 18.

Rule 10 - Basic/Composite Block Exclusivity.

Blocks need to be *either* basic blocks *or* composite blocks. Mixing of textual equations with (graphical) block instances is not allowed.

Rationale: Mixing textual equations with graphical block instances in one block can be very confusing since a modeler may expect that the semantics of a composite block can be entirely deduced from the graphical level.

Trace: Requirement 15.

Rule 11 - Semantical Unambiguousness at Graphical Layer.

The semantics of a composite block must be completely understandable at the graphical layer.

Rationale: This is required since otherwise code reviews can not be done at the graphical level.

Trace: Requirement 12, 15.

Rule 12 - Scalar Signal Extraction via “(De)mux” Only.

Direct scalar signal connections from and to array connectors are not allowed. Intermediate “(De)mux” blocks must be used when scalar signals shall be connected with array connectors.

Rationale: Improves clarity and understandability of models.

Trace: Requirement 12, 15.

4.4 The SAFEDISCRETECONTROL Library

The conceptual SAFEDISCRETECONTROL library provides a restricted set of modeling blocks compliant with the requirements formulated in Section 3. Figure 10 gives an overview about the structure of the library.

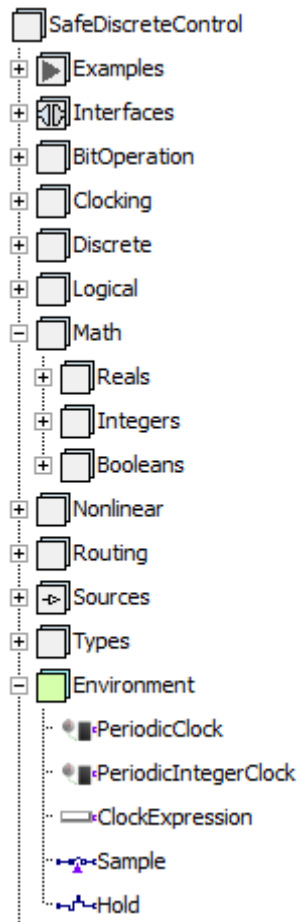


Figure 10: Structure of the SAFEDISCRETECONTROL library. Note that the Environment package contains blocks for modeling the environment in which the modeled high-level application is executed. However, these blocks may not be part of a high-level application (software) model.

As may be expected, the library has to duplicate many blocks found in the Modelica.Blocks standard library. However, it also needs to provide a user friendly access to the elements from the language superset, e.g., extended data types. Figure 11 shows a block for adding two integer signals that includes a choices menu to further specify the integer platform type to be used.

Figure 12 shows a traffic light controller modeled with the SafeDiscreteControl library. The controller is motivated by the example described in [13].

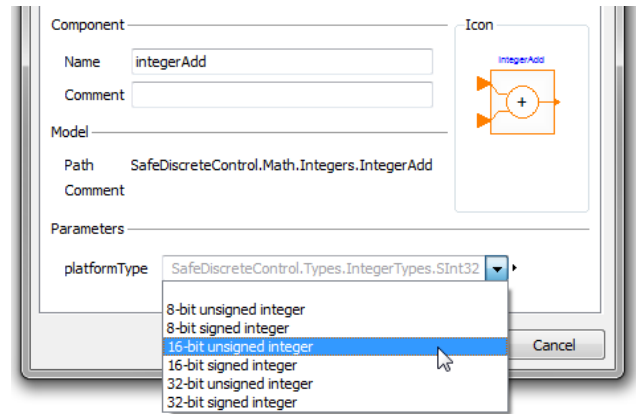


Figure 11: Integer addition block with extended data type support.

The controller's output are the interval lengths of the green phases, respectively for the north-south and the east-west direction. Note that an atom icon in the upper right corner provides the visual information that the block has been declared atomic.

The inside of the controller composite block is depicted in Figure 13. Semantically relevant information like data types or vector valued signals (e.g., between the multiplex and summation blocks) are clearly visible.

Adherence to the modeling rules described in Section 4.3.2 in combination with an adequate library allows to comply with the graphical level requirements formulated in Section 3.2. As a consequence, high-level application development (Role 1), as well as the model reviewing (Role 3), could be done entirely at the graphical level for the presented example.

Please note the presented SAFEDISCRETELIBRARY is a *conceptual* library that was created to check whether it is feasible to model typical discrete control algorithms and their related support logic solely with the use of the proposed sub- and superset of the Modelica language. It is therefore not a library that is available or usable for production purposes!

5 Validation Suites

In order to more clearly understand the role of validation suites in the qualification of development tools, the following section will provide formal definitions of relevant terms and a formalized description of the interplay between development tools and validation suites. Additionally this theoretical framework allows us to specify the relationship between specification

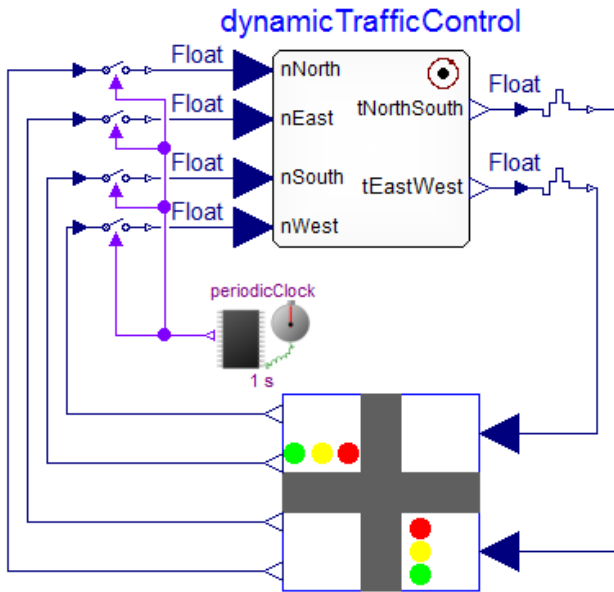


Figure 12: Model of a dynamic traffic light controller including the physical plant model of an intersection of two roads. The controller adjusts its timing and phasing to meet changing traffic conditions. Atomicity of the controller composite block is denoted by the atom symbol in the upper right corner. Data types are visible at the input and output connectors.

models in Modelica and code generation models in the proposed sub- and superset of Modelica.

5.1 Definitions

We define

- M as the set of all valid input models of the intended development tool chain suitable for code generation, i.e. in our case the set of models valid in the proposed Modelica sub- and superset.
- $\tilde{M} \subset M$ as the set of models given by a defined language subset for which the tool chain is to be validated²²,
- $\hat{M} \subset \tilde{M}$ as the set of all test models of a validation suite,
- S_m as the set of all valid stimuli for a given model $m \in M$, and
- $\hat{S}_m \subset S_m$ the set of all test stimuli of a validation suite for a given model $m \in \hat{M}$,

²²Trivially \tilde{M} can be M , though in practice the language is usually further subset to work around known defects in the code generator, elide unused language constructs or avoid language constructs not suitable for the intended application domain.

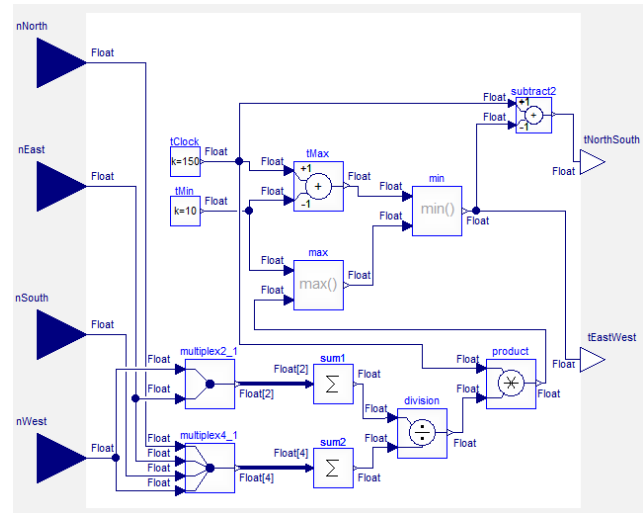


Figure 13: Inside the traffic light controller composite block. All semantically relevant information is visible at the graphical level.

and further the evaluation function

$$\text{eval}_{mil} : M \times S_m \rightarrow R_m$$

of a model by a theoretical simulator (Model-in-the-Loop), where R_m is the set of possible results of a given model $m \in M$.

We are interested in the proper functioning of a development toolchain (DT) consisting of an automatic code generator (ACG), compiler (C), assembler (A) and linker (L) for a target (t), so that

$$DT_t : M \rightarrow B_t = L_t \circ A_t \circ C_t \circ ACG_t$$

is the translation function of a model $m \in M$ into a binary B_t executable on target t .

Due to possible differences in the representations of stimuli and results between host and simulation (S_m, R_m) vs. target and executable (S'_m, R'_m) we also define mappings

$$g : S_m \rightarrow S'_m$$

$$h : R'_m \rightarrow R_m$$

converting between the different representations. Ideally $S'_m = S_m$ and $R'_m = R_m$ and thus $g = \text{id}_{S_m}$ and $h = \text{id}_{R_m}$.²³

Finally

$$\text{eval}_{bil_t} : B_t \times S'_m \rightarrow R'_m$$

is the evaluation function of the binaries $b \in B_t$ on the target t by the processor (Binary-in-the-Loop).

²³The mappings g and h are typically defined as component-wise mappings on the underlying algebraic data types.

5.2 The Task of a Validation Suite

We require from a correct development toolchain that

$$\forall m \in \tilde{M} : \forall s_m \in S_m : \\ d(\text{eval}_{mil}(m, s_m), h(\text{eval}_{bil_t}(DT_t(m), g(s_m)))) = 0$$

for a given metric $d : R_m \times R_m \rightarrow \mathbb{R}$.

Excluding the impact of d it is to be shown that the diagram in Figure 14 commutes.

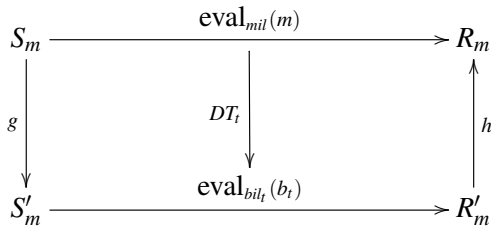


Figure 14: The diagrams shows the interaction of the defined sets and mappings.

Since it is generally not feasible to demonstrate this, the task of the validation suite is to gain confidence in the validity of this assertion by demonstrating the validity of the assertion only for the set of test models and test stimuli, i.e.

$$\forall m \in \hat{M} : \forall s_m \in \hat{S}_m : \\ d(\text{eval}_{mil}(m, s_m), h(\text{eval}_{bil_t}(DT_t(m), g(s_m)))) = 0$$

and ensuring through a suitable selection of the subsets $\hat{M} \subset \tilde{M}$ and $\hat{S}_m \subset S_m$, and additional measures of quality assurance, like e.g. fault-injection, that the generalisation to $m \in \tilde{M}$ and $s_m \in S_m$ is defensible.

5.3 Structure of the Sets M and B_t

The preceding analysis makes no reference to the structure of the sets of valid models M and the set of executable binaries B_t . We will analyse these sets in the following through the lens of the theory of algebraic specifications ([7, 8]).

In this context these sets are the sets of all terms with variables corresponding to their underlying signatures Σ_M or Σ_{B_t} . The sets of stimuli S_m or S'_m are then the sets of all possible values of the variables for given terms in M and B_t .

The evaluation functions eval_{mil} or eval_{bil_t} are correspondingly extended evaluation functions for a given Σ_M - or Σ_{B_t} -algebra, realized by the simulator or the target processor t .

The function DT_t is thus a transformation of terms from $M = T_{\Sigma_M}$ into terms of $B_t = T_{\Sigma_{B_t}}$. A closer examination of the resulting properties of the functions DT_t , g , h , eval_{mil} and eval_{bil_t} , especially with the means of category theory of the categories defined by Σ_M and Σ_{B_t} , and their corresponding algebras can be helpful: The structure of the sets $M = T_{\Sigma_M}$ and $B_t = T_{\Sigma_{B_t}}$ has particular influence on the selection of the test sets \hat{M} and \hat{S}_m , since the structure of these sets also affects the internal structure of the definition of the function DT_t to be tested.

In the test strategy of a validation suite, as described in [14], this observation among other considerations motivates the introduction of specific test areas dealing with the structure of the programming language, e.g. test areas *1 basic constructs of language*, *2 combined constructs of language*, and *4 inner equivalence*, as well as the structure of the transformation process, e.g. *6 internal structure of the code generator*.

Obviously the complexity of the structure, especially the number and kind of different language constructs as well as the number of different ways of combining them and any non-local effects, will determine to a large degree the size of the required test sets. Importantly this relationship due to combinatorial size explosion is at minimum quadratic or cubic, and possibly exponential in complexity. Therefore all effort should be expended to keep the language subset and the complexity of the language semantics as small and simple as possible.

5.4 The Relation between Specification Models and Code Generation Models

We assume in the following that the specification model of Figure 1 is a program in the language Modelica and the code generation model is a program of a possible subset of the proposed sub- and superset of the language Modelica for safety-relevant control applications. Then if we define

- \underline{M} as the set of all valid Modelica models

there exists a mapping $r : M \rightarrow \underline{M}$ for every model $m \in M$ to a corresponding Modelica model $\underline{m} \in \underline{M}$. Thus

$$\tilde{M} = r(\tilde{M})$$

is the subset of valid specification models corresponding to the set of code generation models expressible in the validated language subset.

Conversely, the left-unique, left- and right-total relation

$$p \subseteq \tilde{M} \times \tilde{M} = \{(\underline{m}, m) \in \tilde{M} \times \tilde{M} : r(m) = \underline{m}\}$$

represents the mapping between specification and code generation models.

The reason for p not being right-unique or functional is indicative of the freedom of choice of the developer in their implementation, for example in the choice of implementation data types, which are usually left open in the specification model.

If we restrict this freedom of choice by e.g. using a strict mapping of the data types²⁴ and providing other default implementation choices, we obtain a right-unique relation and with that a bijective mapping function

$$p' : \tilde{M} \rightarrow \tilde{M}$$

with

$$\forall \underline{m} \in \tilde{M} : \exists m \in \tilde{M} : (\underline{m}, m) \in p \wedge p'(\underline{m}) = m \wedge \underline{m} = r(p'(\underline{m}))$$

With corresponding definitions for the functions $g : \underline{S}_m \rightarrow S_m$ and $h : R_m \rightarrow \underline{R}_m$ we expand the diagram from Figure 14 for the evaluation function $\text{eval}_{\text{spec}} : \underline{M} \times \underline{S}_m \rightarrow \underline{R}_m$, see Figure 15.

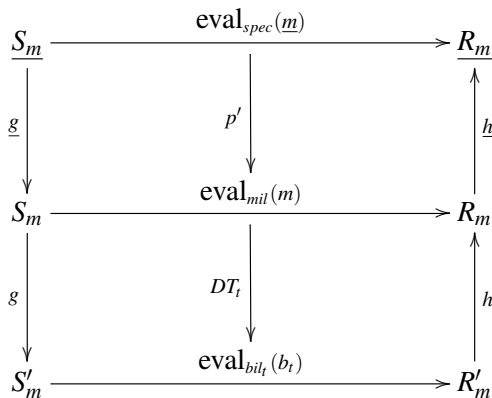


Figure 15: The diagram extends the diagram of Figure 14 with the interaction of the level of the specification model.

This diagram should commute, too, if needed taking into account a suitable (pseudo-)metric $\underline{d} : \underline{R}_m \times \underline{R}_m \rightarrow \mathbb{R}$. For other models $m' \in \tilde{M}$ with $(\underline{m}, m') \in p$ and $m' \neq m$ the diagram in Figure 15 may commute with suitable mappings \underline{g} , \underline{h} and a (pseudo-)metric for suitable stimuli.

Taken together these characteristics make it possible to produce a set of validated tools for both the code generation from code generation models and the transformation from specification to code generation models, so that the probability of fault injection along those two (independent) transformations can be minimized.

²⁴We map e.g. all double in the specification models to double in the code generation model, etc.

6 Conclusion

The article presented a general set of *requirements* that need to be imposed on a high-level, domain-oriented modeling language and its development tools in order to use it for the development of safety-relevant applications. Based on these requirements the suitability of using Modelica within a safety related development process was further analyzed and a *sub- and superset of the Modelica language* was proposed that seems capable of satisfying the formulated requirements.

As a preferred method of choice to gain confidence in software development tools the use of a *validation suite* was proposed and a formal description of the role of a validation suite within a tool qualification effort was given. The precise understanding of the task and effort needed to qualify a tool (based on the validation suite method) is necessary to appreciate the importance of minimizing number and complexity of the allowed language elements.

A prototypical development of a block diagram library (denoted SAFEDISCRETECONTROL) based on established data flow semantics was started in order to test the suitability of the proposed language set to satisfactorily model typical control applications. The first analysis is very encouraging. Currently, there are approximately 60 candidates of blocks and their parameters for the intended SAFEDISCRETECONTROL library. This is comparable to the number of blocks and parameters of already successful validation projects for comparable modeling languages and their development tools, so that the implementation of a validation suite for this language set seems eminently possible.

The next task will be to demonstrate the practical suitability of the proposed sub- and superset (which comprises less than half of the language elements of the Modelica Standard 3.3) for real applications. While it will likely become necessary to enhance or modify the language set based on the practical lessons learned, it will remain of crucial importance, to limit the number of the blocks in the SAFEDISCRETECONTROL library as much as possible:

Experience with validation suites for other modeling languages has shown, that for a language with around 90 basic building blocks, the test sets of the test areas dealing mainly with language structure (i.e. test areas *1 basic constructs of language*, *2 combined constructs of language*, and *4 inner equivalence*, see again [14]) already comprise around 223 000 test outputs. Assuming only quadratic or cubic growth, an increase of a mere 10% in blocks and parameters will result in an increase of approximately 20–30% in val-

validation effort. A strong release management process on the SAFEDISCRETECONTROL library with an ideal limit of about 50 basic building blocks therefore seems advisable.

It should be noted that tool qualification is most effective when performed in an early phase of the development process: Errors in development tools are usually hard to detect and analyse in normal development, and the effects of work-arounds and tool limitations can have a huge impact on the efficiency of the development process when introduced at a late stage of development. Tool qualification should therefore be concluded prior to the start of the development project. Ideally, tool qualification efforts start even before the language definition is finalized: When language constructs and definitions are viewed through the lens of tool qualification, error-prone or hard to test constructs and corner cases are highlighted, and improvements can still be adopted. By the concurrent creation of a test suite, interpretation of the language definition can be clarified and harmonized between possible implementations. For this reason we would welcome further work in this area even at this early stage of language set definition.

The authors hope that, on the one hand this contribution sheds some light on the requirements and intricacies that need to be faced when considering the usage of Modelica for safety related applications, and on the other hand hope to stimulate further discussion on the rationale and possible approaches to employing Modelica in this field.

Acknowledgments

The first author would like to thank Dominik Sommer for the valuable discussions about safety relevant software development in aerospace applications.

References

- [1] Albert Benveniste, Stephen A. Edwards, Nicolas Halbwachs, Paul Le Guernic, and Robert de Simone. The synchronous languages 12 years later. In *Proceedings of the IEEE*, volume 91 (1), pages 64–83, 2003.
- [2] Dariusz Biernacki, Jean-Louis Colaço, Gregoire Hamon, and Marc Pouzet. Clock-directed modular code generation for synchronous data-flow languages. *SIGPLAN Not.*, 43(7):121–130, June 2008.
- [3] Tilman Bünte, Akin Sahin, and Naim Bajcinca. Inversion of Vehicle Steering Dynamics with Modelica/Dymola. In Gerhard Schmitz, editor, *4th Int. Modelica Conference*, March 2005.
- [4] David Broman. *Meta-Languages and Semantics for Equation-Based Modeling and Simulation*. PhD thesis, Linköping University, PELAB - Programming Environment Laboratory, The Institute of Technology, 2010.
- [5] Manfred Broy, Helmut Krcmar, Jens Zimmermann, and Sascha Kirstan. Einfluss des Software-Designs auf die Wirtschaftlichkeit von Software-Entwicklungen. *ATZelextronik*, 02:34–37, April 2011.
- [6] 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.
- [7] Hartmut Ehrig and Bernd Mahr. *Fundamentals of Algebraic Specification 1: Equations and Initial Semantics*. Monographs in Theoretical Computer Science. An EATCS Series. Springer, 1985.
- [8] Hartmut Ehrig and Bernd Mahr. *Fundamentals of Algebraic Specification 2: Module Specifications and Constraints*. Monographs in Theoretical Computer Science. An EATCS Series. Springer, 1990.
- [9] IEC 61508: Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems, 1998.
- [10] Modelica Association. Modelica—A Unified Object-Oriented Language for Systems Modeling v3.3. Standard Specification, May 2012. available at <http://www.modelica.org/>.
- [11] Ramine Nikoukhah and Sébastien Furic. Towards a full integration of modelica models in the scicos environment. In *7th Modelica Conference, Como, Italy*, September 2009.
- [12] Constantinos C. Pantelides. The consistent initialization of differential-algebraic systems. *SIAM Journal on Scientific and Statistical Computing*, 9(2):213–231, 1988.
- [13] Stefan-Alexander Schneider and Tobias Hofmann. Functional Development with Modelica: A Use-Case Analysis. In *9th Int. Modelica Conference*, Munich, Germany, September 2012.

-
- [14] Stefan-Alexander Schneider, Tomilav Lovric, and Pierre Mai. The validation suite approach to safety qualification of tools. *SAE Technical Paper 2009-01-0746*, 2009.
- [15] Robert Tarjan. Depth-first search and linear graph algorithms. In *Switching and Automata Theory, 1971., 12th Annual Symposium on*, pages 114–121, oct. 1971.
- [16] The Motor Industry Software Reliability Association. MISRA-C:2004 - Guidelines for the use of the C language in critical systems, 2004. <http://www.misra.org.uk>.
- [17] The Motor Industry Software Reliability Association. MISRA AC AGC - Guidelines for the application of MISRA-C:2004 in the context of automatic code generation, 2007. <http://www.misra.org.uk>.
- [18] M. Thümmel, M. Kurze, M. Otter, and J. Bals. Nonlinear inverse models for control. In *4th Int. Modelica Conference*, pages 267–279, 2005.
- [19] Michael Thümmel, Martin Otter, and Johann Bals. Vibration control of elastic joint robots by inverse dynamics models. In H. Ulbrich and W. Günthner, editors, *IUTAM Symposium on Vibration Control of Nonlinear Mechanisms and Structures*, pages 343–353, München, 2005.

A Modelica Library for Industrial Control Systems

Marco Bonvini Alberto Leva
Politecnico di Milano, Dipartimento di Elettronica e Informazione
Via Ponzio 34/5, 20133, Milano, Italy

Abstract

Many studies for which simulation is necessary include the presence of control systems. While plenty of Modelica libraries are nowadays available to accurately represent the plant, the same is not so true as for the control elements, since industrial ones are endowed with a number of functionalities – and often present system- or even vendor-specific peculiarities – that are not represented by the typical blocks (e.g., based on transfer functions) offered by the existing libraries. This paper is an attempt to start filling the gap and provide an efficient solution, structured and organised in such a way to be easily understood by control specialists, and to ease information transfer between simulation studies and implementation.

Keywords: industrial controllers, simulation

1 Introduction

In many simulation studies, control plays a relevant role. Sometimes this is because the study is precisely aimed at setting up the control system for the plant at hand, but in many other situations, even if control synthesis is not the main goal of the study, the behaviour itself of the modelled object depends significantly on the operation of some controls. As such, quite often the representation of the control system deserves substantially the same accuracy as the representation of the physical plant (in the broad sense of the term).

At present, numerous Modelica libraries are available to represent plants with a virtually arbitrarily accuracy, but the same is not true – at least, to the best of the authors' knowledge – for controllers. To appreciate that, the interested reader could for example throw a glance at the PID block as provided by any control environment, be it targeted to a PLC, a DCS, or whatever. Most likely, he/she will see something similar to the two examples shown in figure 1.

Apparently, such blocks are more articulated than for example the PID of the Modelica Standard Library

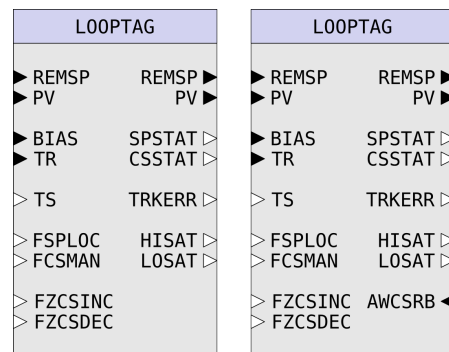


Figure 1: Two examples of PID blocks as seen in typical industrial control tools.

(MSL)—as by the way real-life control systems do exhibit a number of peculiarities that are not accounted for in “textbook” representation, see e.g. [9]. The remarks just made are in no sense meant to be a criticism, it is worth stating; nonetheless they evidence that for the simple controller representations of the MSL (or analogous ones) to be adequate, some conditions are necessary. Summarising, and sticking to the PID example,

- the specific form of the controller (let alone the detailed operation of the control algorithm) must not be relevant for the problem,
- and the operation of typical elements of industrial controllers, such as tracking and locks, must not be of concern either.

If this is the case, MSL-like representations are perfectly adequate. If on the contrary either this is not the case in the simulation *scenarii* to be considered, or one wants to describe the control system so as to be capable of simulating the controlled plant in its entire set of operating modes, the same representations cannot serve the desired purpose.

For the reasons above, and after several years during which the authors and their group have been developing *ad hoc* solutions for individual cases, the decision

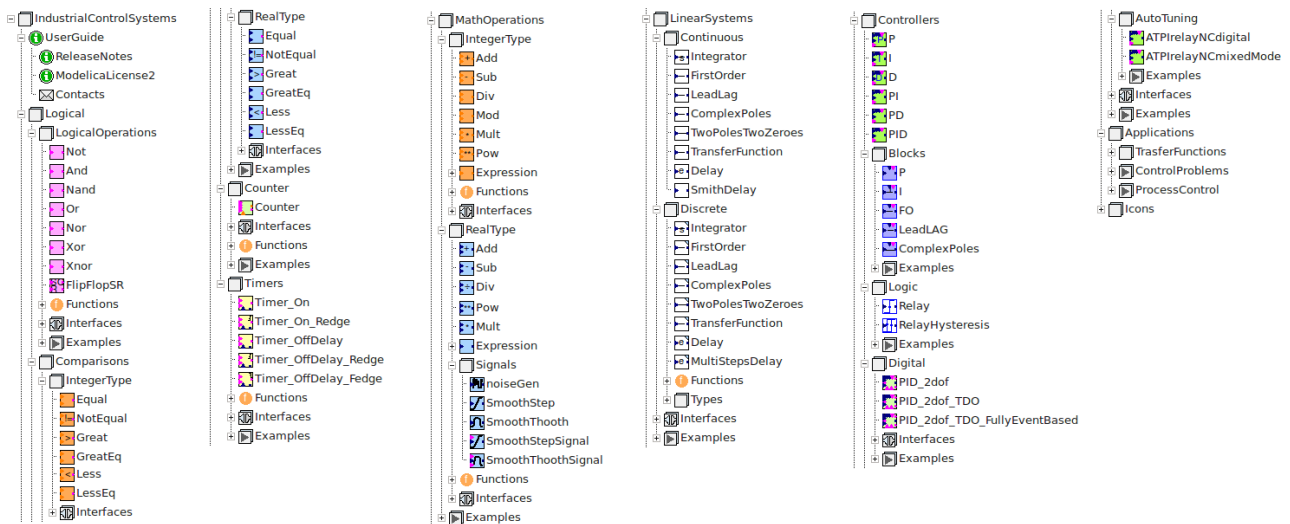


Figure 2: An overview of the library structure.

was recently taken to put all of that knowledge and Modelica code together in a structured manner.

The result is the library described in this paper, which is organised as follows. Section 2 presents and motivates the most qualifying characteristics of the library. Section 3 presents the library structure giving just a quick overview, as the library documentation provides full detail on the matter. Section 4 reports some simulation examples, these too available in the library, to evidence and further motivate its distinctive characters previously discussed. Finally, in section 6 some conclusions are drawn, and future work is sketched out.

2 Main characteristics of the library

To fulfil the requirements envisaged in section 1, it is first necessary to include both modulating and logic control elements.

For modulating elements, it is required to account for the typical representations of the major control blocks – see e.g. [8, 3] for how many forms a PID can take – and the typical realisations of the main nonlinear functionalities: for example, taking again the PID as example, antiwindup can be realised internally or by reading back the applied control from the actuator. Also, logic functionalities need incorporating, such as tracking and the possibility of preventing the control signal from increasing or decreasing, which is of great usefulness in cascade controls. Finally, different algorithmic realisations (e.g., positional or incremental) need considering, since in some cases they can affect

the behaviour of the element, especially if controller parameters can be modified online as is the case for gain-scheduling blocks.

For logic elements, the typical set available in SCADA-like products needs representing, including timers, counters, sequencers, and so forth.

Then, it would be advisable that the modelled control elements allow for variable-step simulation, to avoid obliging the analyst to use the library only with fixed-step solution, which could be unacceptably inefficient in more than one case. As such, the choice was made to provide both a time-driven and an event-driven version of the same element wherever this is possible, and research is underway to extend this coverage to the whole library.

Moreover, in a view to good acceptance and wide utilisation, care was taken to give the library elements a look and feel as similar as possible to what a user of SCADA (or analogous) tools expects to see. This was not pursued up to its extreme consequences, but is definitely a peculiarity.

Finally, an initial set of autotuning controllers is included, building on previous research see e.g. [1, 5, 7, 6]; this is meant both to ease control setup in simulation, and to help the user familiarise with that technology, and the underlying theory.

3 The library structure

The library is organised into subpackages; a list of the major ones is given below.

- **Logical**, that contains all logical elements, timers, counters, and so forth.
- **MathOperations**, including the necessary operators for real and integer numbers (which is sometimes very useful to correctly represent the operation of some industrial blocks).
- **LinearSystems**, where some blocks are contained that can be used to easily close loops to test controllers. Part of those blocks are also related to well known controller benchmarks, see e.g. [2]; of course this subpackage is provided basically for convenience and to obtain a self-contained library, but many alternatives can be used.
- **Controllers**, where both modulating and logic control blocks are represented, in three basic (and interchangeable) manners: (a) as continuous-time equations, (b) as equations but evolving by events, and (c) – when multiple assignments could not be avoided, although research to solve this is underway – as algorithms.
- **Applications**, that contains a quite large set of examples to better understand and use the library.

Figure 2 shows an overview of the library structure. Readers that are familiar with control systems and control theory will easily get familiar with the library and its structure (just by observing the library components); non experienced user will find further details into the included documentation.

3.1 Interfaces

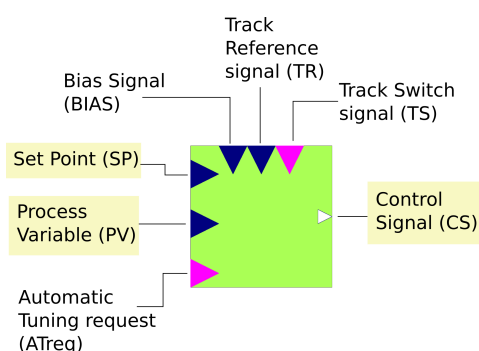


Figure 3: Interface for a generic controller. The input/output connector evidenced in yellow are always present, the other ones can be conditionally selected.

Each model/block/controller contained into the Industrial Control Library can be connected together

Table 1: This table contains the definition of the interface of a generic controller with its conditional input/output connectors.

Name	Description	Conditional?
SP	Set Point	NO
PV	Process Variable	NO
CS	Control Signal	NO
TR	Track Reference	YES
TS	Track Signal	YES
Bias	Bias signal	YES
ATreq	Automatic Tuning request	YES

with other models ones through its standard connectors, defined in the Modelica Standard Library. In each subpackage, an ad-hoc partial interface model has been defined in order to improve the readability of the code, and reduce as much as possible the number of code lines spent for non specific purposes. Figure 3 shows the interface of a generic controller. The input/output connectors of such a block can be conditionally selected through various boolean flags as shown in table 1. With these conditional connectors a controller can be used even if it does not use all its features, without connecting dummy inputs to it and thus increasing the clarity of the control scheme. The interfaces and the variables of the models have been named according to the standard terminology in the field of control systems. The interested reader that is not familiar with this topic can find more information in [4].

4 Simulation examples

This section contains a small sample of the examples contained in the library, to show the possible usage of some models, and also evidence the usefulness of adopting the proposed representation.

4.1 Zero crossing count

This examples uses some blocks of the Logical subpackage, as shown in figure 5. More in detail, the signal represented by $y(t) = \sin(t)$ is compared with to $z(t) = 0$. Each time the signal crosses the reference, the boolean output of the comparison block rises. The rising edges are counted by the digital counter, in the period comprised between $t = 2.2$ and $t = 10.2$. Figure 5 reports the Set and Reset Count signals, while figure 6 shows the behaviour of the counter value.

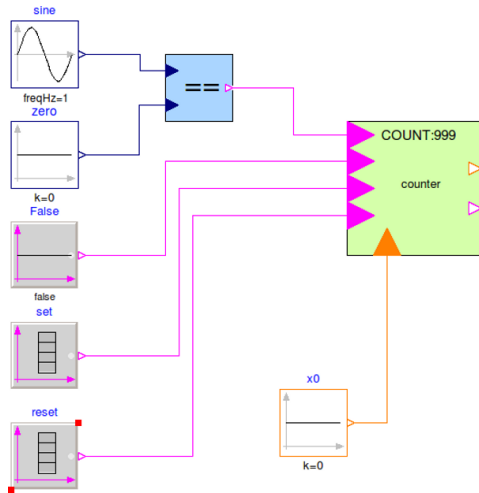


Figure 4: Scheme of the zero crossing count model.

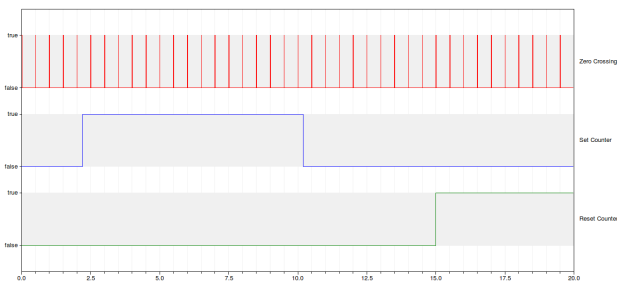


Figure 5: Zero crossing signal, Set count signal and Reset count signal.

4.2 PID with bias and tracking mode

In this example the second order process defined as

$$P(s) = \frac{(1 + 15s)}{(1 + 2s)(1 + 10s)} \quad (1)$$

is controlled by a PID regulator, to track given step Set Point signal, and reject a load disturbance acting on the process input (as shown in figure 7). Two controllers are compared, namely a PID and a PID with bias input. Figure 8 reports the Set Point (blue line), the Process Variable of the process without control (red), controlled with the PID (green) and the PID with bias

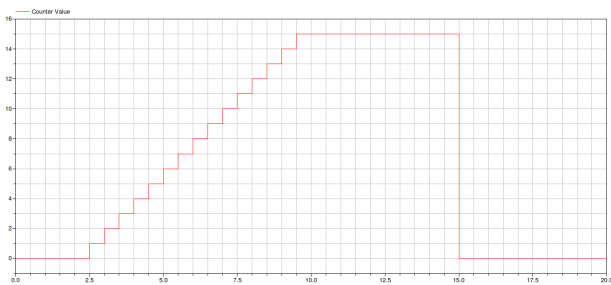


Figure 6: Counter value.

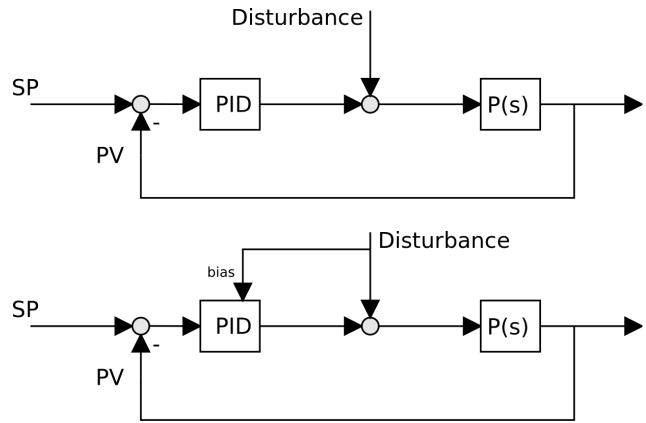


Figure 7: Classic PID controller: without bias signal (top) and with bias signal (bottom).

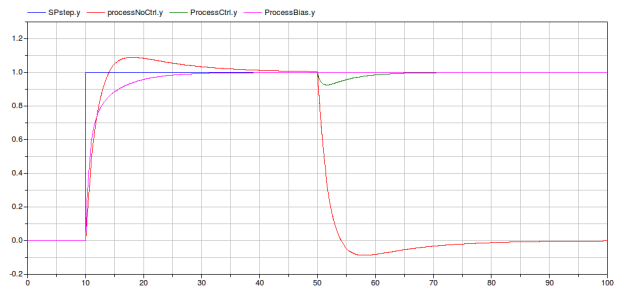


Figure 8: Set Point (blue), Process Variable without control (red), with a PID controller (green) and with a biased PID controller (magenta).

(magenta). The PID rejects the disturbance just via the feedback path, that makes its action slower. On the contrary, the PID with bias acts immediately, thanks to its feed forward character.

Carrying on to representing the tracking mode operation (see figure 9), an example is shown with the process defined in (1), still controlled by a PID. Figure 10 shows the Set Point, the process variable and the tracking switch signal, while figure 11 shows the control signal, the track reference and the integral action of the controller. The Tracking mode starts at $t = 40s$, before the controller has led the process variable to the Set Point reference. When the Tracking mode starts, the control signal becomes equal to the track reference (as shown in figure 11). In this case the track signal decreases and then increases, moving the process variable in a neighbourhood of the set point. When the tracking mode is enabled, the integrator does not integrate the error signal, rather is managed in such a way to be consistent with the track reference. Thus, the transition from the tracking mode to the automatic one is bumpless.

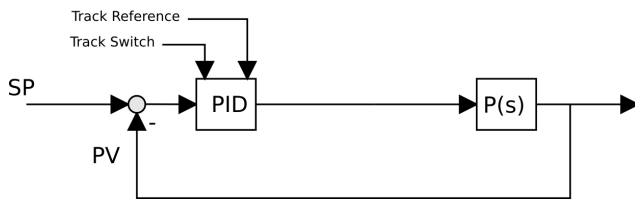


Figure 9: PID with Track Switch and Track Reference signals

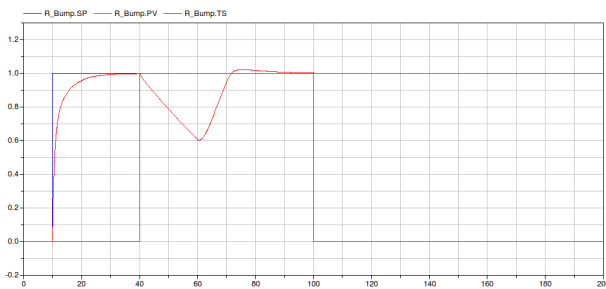


Figure 10: Set Point (blue), Process Variable (red), and Track Switch signal (green).

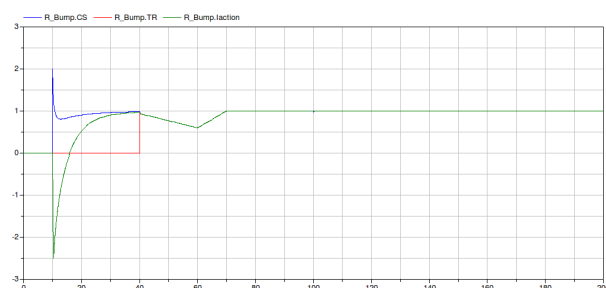


Figure 11: Control Signal (blue), Track Reference Signal (red) and Integral action (green).

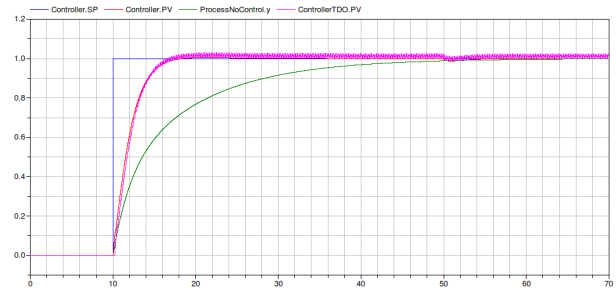


Figure 12: Set Point (blue), Process Variable without control (green), process Variable with PID (red) and PV with TDO PID (magenta)

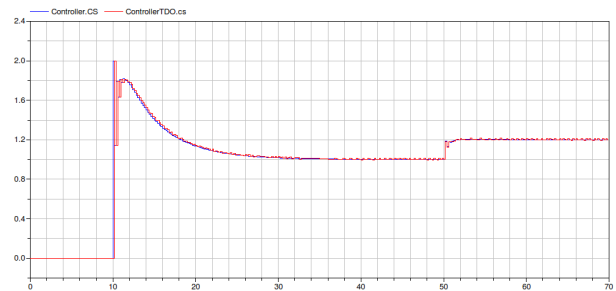


Figure 13: Control Signal (blue) and CS of TDO PID (red)

4.3 Time Division Output controller

The process (1) is here controlled with a Time Division Output PID. Such an actuation scheme is used to have an on/off actuator behave like a modulating one, and is quite typical when either the actuator cannot be partialised, or doing so would unacceptably reduce its efficiency. The controller, implemented in its digital algorithmic form, first computes the control signal, and then converts it into the duty cycle of a rectangular wave of assigned period. Figure 12 contains the Set Point reference (blue), the Process Variable of a process without control (green), the Process Variable of a digital PID (red) and the Process Variable of a TDO PID (magenta). Since the TDO control signal changes continuously, the relative process variable has a sort of ripple, however the overall behaviour is essentially the same as the digital PID without TDO. The control signals computed by the two controllers are shown in figure 15.

4.4 Cascade control with increment and decrement locks

This examples compares two cascade control schemes, one with and one without increment/decrement locks. When two controllers are connected together in a cas-

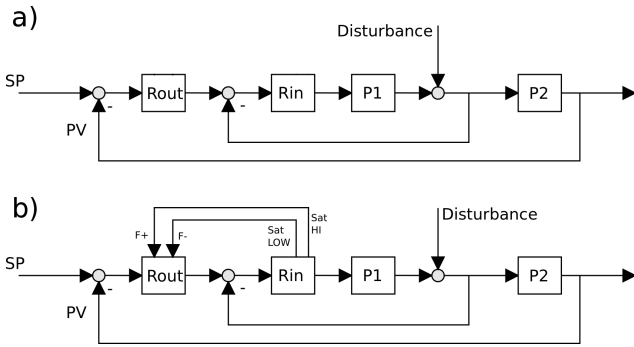


Figure 14: Cascade control schemes: a) without increment/decrement locks – b) with increment/decrement locks.

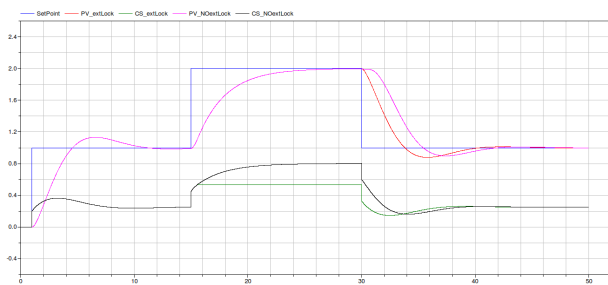


Figure 15: Cascade control with and without increment/decrement locks – outer set point and process variable, inner set point.

cade control scheme, the inner controller typically regulates the actuator, while the outer one provides the Set Point reference for the inner one. Since the inner controller acts on the plant, its Control Signal has to be limited, and AntiWindup is in order, but in general it is not possible for the *outer* controller, to know the values for which the *inner* regulator saturates.

Such a problem can be avoided by using the PID in its incremental form, using the Increment/Decrement lock feature, and creating an external (logical) loop between the controllers, as shown in figure 14.

If the inner regulator saturates, its satHi signal becomes true. Connecting this signal to the forbidIncrement input of the outer controller, avoiding a useless and potentially dangerous increase of its Control Signal (that is the Set point of the inner controller that saturated). With such a scheme, the mentioned inter-loop windup-like effect can be avoided.

In figures 15 and 16, that show the results, the green line is the CS of the outer controller with Increment/Decrement lock, while the black one is the output of the outer controller without Increment/Decrement lock. The black line shows a windup like effect that turns in a slower reaction when the Set

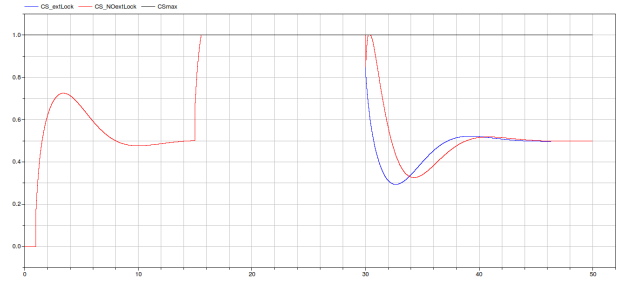


Figure 16: Cascade control with and without increment/decrement locks – inner control.

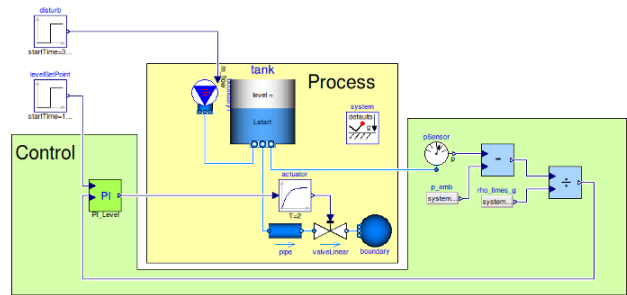


Figure 17: Modelica diagram of a level control scheme. The two subsystems (the control system and the process to be controlled) are evidenced with different background colors.

Point changes at time $t = 30$.

4.5 A level control case

In this example, models from the presented library are used together with models from the MSL. The aim of this example is to show the usefulness of the presented models, and how they can be easily integrated and connected with others. For this purpose, the chosen example refers to the problem of controlling the water level in a tank. The water level is the process variable, and the system (see figure 17) is composed of a tank and a pipe connected to a valve, that discharges water to the atmospheric pressure. The valve actuator is simply represented by a first order system with unity gain.

The control system is composed of a measurement part and a control (*stricto sensu*) one. Concerning the measurement part, the pressure sensor measures the absolute pressure at the bottom of the tank. The measured pressure p_m is subtracted from the atmospheric pressure p_0 , and then divided by the gravity acceleration g and the water density ρ , in order to obtain the water level

$$l = \frac{p_m - p_0}{\rho g} \quad (2)$$

The PI controller, given the level measurement and

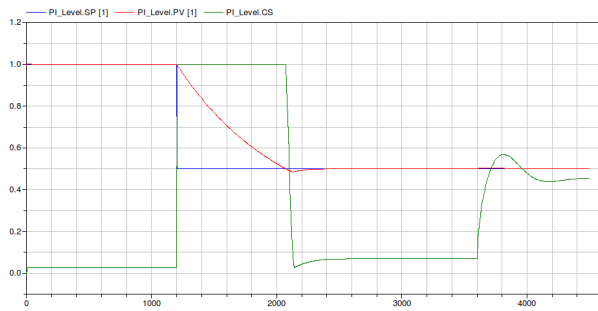


Figure 18: Set Point water level reference, Process Variable and valve position command

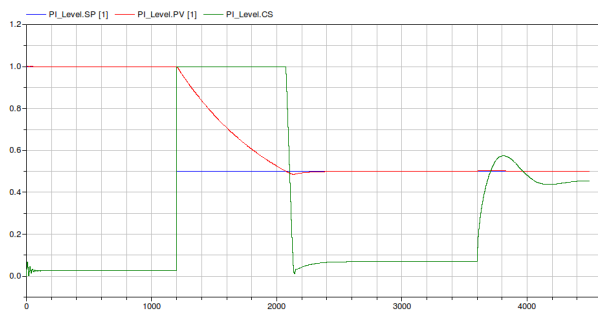


Figure 19: Set Point water level reference, Process Variable and valve position command (discrete time controller, $T_s = 5$ s)

the set point, computes its control action, i.e., the prescribed valve position, limited between $CS \in [0, 1]$ in order to avoid windup effects (thus $CS_{min} = 0$ and $CS_{max} = 1$). The tank is $2m$ height, and the water level at time $t = 0$ is $L = 1$ m. In the first phase the controller is required to maintain the level at the initial value ($SP = 1$ m), while at $t = 1200$ s the level set point has a steep variation ($SP = 0.5$ m). The controller has to act on the valve in order to decrease the water level to the desired value. A disturbance, represented by a water mass flow rate entering the tank, becomes different from zero at time $t = 3600$ s. Figure 18 shows reference, water level and valve position command.

The simulation can be performed at an initial stage assuming that the controller is a continuous time one ($T_s = 0$), and the math operations are in double precision (FixedPoint = false). In such a phase, it is thus possible to concentrate on the controller design (not on implementation-related facts).

As a further stage, one could introduce more details in order to simulate a more realistic system. At first it is possible to introduce the time discretisation, and investigate the effects of the sampling time. Figure 19 shows the simulation results with a sampling time $T_s = 5$ s.

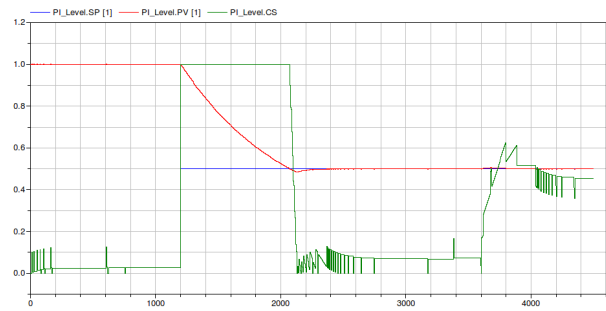


Figure 20: Set Point water level reference, Process Variable and valve position command (discrete time controller, $T_s = 5$ s and Fixed Point math operations)

An additional level of detail can be the introduction of fixed point math operations. In this example, a number of bit $N_{bit} = 24$ was chosen, which means that the integer number that can be represented are comprises between $MIN = -8388609$ and $MAX = 8388608$. At a first stage, the measured pressure have to be subtracted of the ambient one. In the worst case, the higher pressure value that can be read as input from the math operation block is $101325 + 1000 \cdot 9.81 \cdot 2 = 120945$, that is more or less two orders of magnitude less than the higher integer number MAX . This means that input numbers can be multiplied by a scale factor comprised between 10 and 50. In this case the scale factor has been chosen as $sFactor = 20$. In a similar way, the scale factor for the division can be chosen (In this case, $sFactor = 500$). Note that a large number of bits is required because the pressure variation is small with respect to its absolute value. Using such a modelling approach, it is possible to estimate the amount of bits required, and to directly test the correctness of the design strategy. Figure 20 shows that the numerical errors due to a wrong design are visible on the Control Signal.

5 Towards Modelica 3.3

The recent definition of the version 3.3 of the Modelica language introduces new elements for describing synchronous behaviours, and also new elements suited to define synchronous state machines. This evolution is primarily made to ease the activity of modelling realistic control algorithms.

These evolutions will introduce some advantages in the development of models that are pure discrete or logical, since a standardised framework for developing such models will help in the design, creation and maintenance of models in which many of these com-

ponents are connected together. Considering elements that can be either continuous time or discrete time, and which events are not regular but can be dynamically driven; however, it is not yet clear if this language evolutions will fit also such model characteristics, that (as shown) are of great importance for tailoring the simulation burden to the needs of the addressed study.

The last interesting point that has not yet been considered, but in the authors' opinion should be, is the introduction of the Fixed Point arithmetic. The presented library takes into account this problem and it is managed in a preliminary and simplified way, providing a solution just for simple cases. The introduction of a new type of variable with its specific operations will be an important step in the direction of a really control (and control synthesis) oriented simulation tool.

6 Conclusions and future work

A Modelica library for industrial controllers was presented, with several peculiar features, and some examples were shown to illustrate its potentialities.

In the authors' opinion the library can significantly help the analyst who has to address studies where a precise control representation plays a relevant role—more frequent a case than one may expect at a first glance, by the way. The presented library in the first place responds to such a demand, and in addition tries to preserve the advantages of variable-step simulation when possible—a matter on which further research is however underway. The library is by definition extensible, so that one may even want to include the exact (i.e., code *replica*) representation of some block of interest, employing those already realised as a starting point. Implicitly, then, the library has also a didactic value, since the user can see how several concepts are actually put to work. Some examples were reported to show the library operation. All of these – plus others omitted here for space reasons – are available in the library itself (available at <http://home.dei.polimi.it/leva/download.html>), for the convenience of the interested reader.

Future activity (apart from the already mentioned one related to simulation efficiency) will be directed at expanding the library in all its sections, including the autotuning one, and to extensively use it in simulation studies. The community is encouraged to use, improve – and correct if necessary – the library, and feedback would be highly appreciated by the authors in order to continuously improve the results.

References

- [1] K.J. Åström and T. Hägglund. Industrial adaptive controllers based on frequency response techniques. *Automatica*, 27(4):599–609, 1991.
- [2] K.J. Åström and T. Hägglund. Benchmark systems for PID control. In *IFAC Workshop on Digital Control – Past, present, and future of PID Control*, Terrassa, Spain, 2000.
- [3] K.J. Åström and T. Hägglund. *Advanced PID control*. Instrument Society of America, Research Triangle Park, NY, 2006.
- [4] W. Dunn and W.C. Dunn. *Fundamentals of industrial instrumentation and process control*. McGraw-Hill Professional, 2005.
- [5] A. Leva. PID autotuning algorithm based on relay feedback. *IEE Proceedings-D*, 140(5):328–338, 1993.
- [6] A. Leva and M. Bonvini. Efficient hybrid simulation of autotuning PI controllers. In *Proc. 8th International Modelica Conference*, Dresden, Germany, 2011.
- [7] A. Leva, S. Negro, and A.V. Papadopoulos. PI/PID autotuning with contextual model parametrisation. *Journal of Process Control*, 20(4):452–463, 2010.
- [8] A. O'Dwyer. *Handbook of PI and PID controller tuning rules*. World Scientific Publishing, Singapore, 2003.
- [9] F.G. Shinskey. Process control: as taught versus as practiced. *Industrial & Engineering Chemistry Research*, 41(16):3745–3750, 2002.

Modelica3D - Platform Independent Simulation Visualization

Christoph Höger¹, Alexandra Mehlhase¹, Christoph Nytsch-Geusen², Karsten Isakovic³, and Rick Kubiak³

¹*Technische Universität Berlin*

²*Universität der Künste Berlin*

³*Fraunhofer FIRST*

Abstract

Modelica3D is a platform-independent, free Modelica library for 3D visualization. Its implementation is based on a message-passing architecture. Through its loosely-coupled architecture, Modelica3D can be combined with different rendering-tools. It is also highly extensible and scalable.

Keywords: 3D Graphics, Library, Platform Independence, Free Software, Structural Dynamics, Loose Coupling, Message Passing

1 Introduction

Simulation results in Modelica are usually visualized using two-dimensional plots. System states are shown as functions over time or each other. While this is of course a very natural way to approach the presentation of simulation results, it is not sufficient in some aspects:

In Modelica, simulation-models are composed of reusable software fragments. Thus an interesting quantity might not be present directly, but in form of a relation between multiple system states (e.g. distances between different objects). The solution is to either change the model, post-process the simulation results, or to switch to a more complex visualization method.

Additionally, a simulation might be used to drive interactive real-time simulators (e.g. for training purposes) or to present certain facts to a non-simulation audience. In both cases the usage of 3D graphics might lower the barrier significantly for team members, which are not as familiar with the simulation as the responsible engineer. Therefore it is important that the visualization aspects can be controlled from within the simulation environment. On the other hand visual-

ization experts (and expert-tools) are necessary to create realistic and usable 3D-graphics. The Modelica3D library aims to provide a solution for these requirements.

1.1 Contribution

In this paper we will demonstrate how a visualization library (called Modelica3D) can be implemented by using only standard Modelica features. The library itself is available under a free software license. We will show how a tight coupling between the library and the rendering-tool can be avoided. This loose coupling allows the visualization of structural dynamic systems. Additionally, we show how the underlying message-exchange API makes the Modelica3D API both flexible and extensible. Finally, an example is given, demonstrating the scalability of Modelica3D to industrial models.

Finally, the method proposed here is not limited to 3D-graphics. The same means could be used to control e.g. sound output or any other simulation feedback. In that way, Modelica3D demonstrates how simulations can control *effects* beyond their simulation environment.

The rest of the paper is organized as follows: First, we will discuss the state-of-the-art of 3D visualization in the Modelica ecosystem. Then, a technical overview over Modelica3D's architecture is presented. This includes a discussion of the overall design as well as solutions to overcome some Modelica-specific limitations. Second, we will show how Modelica3D can be used to simulate an existing library (Modelica.MultiBody) to achieve tool-independent state-of-the-art visualization. As a second use-case a recently developed technique for finite-state structurally dynamic systems is extended with Modelica3D visual-

ization. Finally, we will evaluate the library by implementing a large-scale industrial model visualization.

2 State of the art

During the last 10 years different approaches were tested to integrate scene descriptions of 3D bodies in the Modelica language or to support 3D visualizations by the Modelica simulation tools.

The first fundamental analysis and conceptual work in this field was done by Engelson [3]. Two alternative ways were discussed for the integration of 3D object information in Modelica: First, the definition of a basic set of “graphical” Modelica classes, which make a representation of primitive 3D objects (e.g. triangle, sphere) and position operations with this objects (e.g. translation, rotation) in user defined physical models possible. Second, the direct integration of the 3D object information as “graphical annotations” into the physical models self.

Another approach of a annotation concept for the embedding of 3D geometries in Modelica was developed from [5], where specialized 3D annotations for model classes and objects and a standardized description of 3D geometries and the related body topologies (in this case the X3D standard) were combined. Within the tool specific approaches, individual ways for the 3D information integration were done by the software developers. The greatest disadvantage consists in the incompatibility of the 3D models, caused by the use of vendor specific 3D information.

The simulation tool SimulationX from ITI supports both for his own Modelica libraries and also for user written Modelica libraries the visualization and animation of 3D objects. With the help of an 3D editor tool, the 3D information is stored in the physical Modelica models and also in related non standardized annotations. The 3D editor supports the definition of simple and complex bodies, which are constructed by the combination of standard 3D primitives and also specialized objects such as gears and spiral springs.

The simulation tool Dymola from Dassault Systemes supports for selected Modelica libraries the visualization and animation of 3D-objects, mainly for the MultiBody-Library. For this, specialized visualization classes for 3D primitives were introduced. Further, complex 3D geometries, based on external definitions of 3D-shapes via dxf-files are utilized. The MultiBody package of the Modelica Standard Library uses data structures defined in Modelica.Services to calculate a *complete* continuous time model of the 3D

visualization geometry. This approach does not allow for *effect-events* (e.g. deformations, material changes).

The visualization framework SimVis for 3D modelling and simulation with Modelica was developed by the German DLR [2]. On the modelling side, a new developed ExternalDevices-library represents the base for the 3D visualization and interactivity. For the simulation experiments three different types of input devices (keyboard, joystick, 3D space mouse) supports the direct 3D interaction by the user. The technical base on SimVis is OPENGL and OPENSceneGraph. Different use cases were analysed within SimVis such as flexible body simulation, energy flow simulation, Head-Up-Display simulations, hybrid cars and robot simulation.

3 Modelica3D

In this section we discuss the architecture of Modelica3D and the design decisions that lead it. As Modelica3D is a purely non-physical library, there are no modeling concerns (e.g. reusable and understandable components) that need to be addressed. Instead, Modelica3D focuses solely on effects *outside* of the simulation. Thus, we could focus on general software design principles and the goals motivated earlier.

3.1 Design Decisions

First of all, Modelica3D should be platform independent: Only methods that are part of the Modelica Specification [1] should be used. This rules out the development in form of an extension to an existing platform and a solution based on vendor-specific annotations. Any tool that follows the specification should be able to use Modelica3D directly. During the development we used OpenModelica [4].

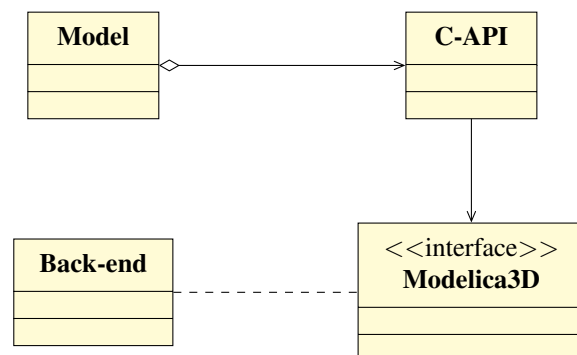


Figure 1: Modelica3D architecture

Instead, Modelica3D must be shippable as a library.

This library must contain a layer of Modelica-Code, which allows access to the 3D API from any model. On the other hand, extensions which cannot be expressed in Modelica need to be implemented in a language that is supported through Modelica’s external function interface. Since only C and Fortran are currently specified, C is a natural choice, being the “lingua franca” of platform independent development.

The second important requirement was loose coupling between front-end (the simulated model) and back-end (the rendering-tool). While with the choice of C as implementation language, several options for accessing rendering-tools exist, directly linking the back-end would cause several drawbacks:

- Only few back-ends can be used as a library. Even if they do (e.g. OpenSceneGraph), the viewer usually requires a lot of additional features (user interface, inputs, file management). Providing those features to a Modelica model in a platform independent implementation would require lots of additional work for each back-end and thus only allow very few implementations.
- A fixed C API would not only put an additional burden on developers who want to extend the library. It would also hinder the maintenance, since every back-end would effectively require it’s own C-library (including it’s own bugs). Ideally the parts written in C should be as small as possible instead, leaving the lion’s share of work to Modelica and back-end experts.
- Linking works only locally. In times of distributed computing it seems unreasonable to demand simulations running on the same physical machine as visualization.

So instead of directly linking 3D API functions into a Modelica model, we chose to use interprocess communication (IPC). That way, front-end as well as back-end can run as dedicated processes while sending respectively receiving messages. Any back-end needs to implement a common *interface* (which is simply the set of messages accepted).

Because visualization should not influence the simulation results, the communication between front- and back-end is *unidirectional*. In our design this allows a further simplification of the message-exchange protocol: Since the front-end does not expect any messages from the back-end, the communication can work *synchronously*. This also fits into the event-driven modeling style of Modelica. Note, that by using *time-events*

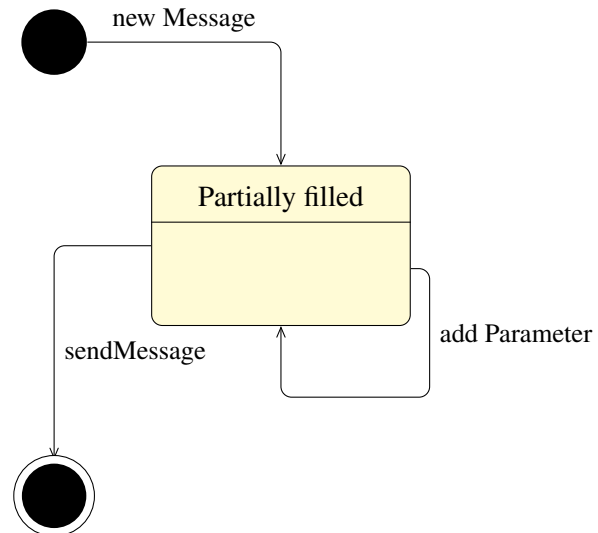


Figure 2: modbus message lifecycle

for the event handling, the effect on the simulation performance can be minimized by the simulation tool, as discussed e.g. in [8].

3.2 Implementation

With the design decisions settled, the first task was to implement an *extensible, synchronous, platform-independent* IPC layer in Modelica. Instead of reinventing the wheel, we chose to use an existing IPC solution and wrap around it’s C-interface. Because of it’s availability, maturity, and simple C-API, the choice fell on dbus, the current de-facto standard for IPC on Linux [7] ¹.

The first part of Modelica3D is thus a thin Modelica wrapper around dbus, called *modbus*. Modbus allows creation and sending of arbitrary messages as *External Objects*. Message objects can be allocated, equipped with parameters and send over a *connection*.

Since modbus only uses very few of features of dbus (only one-to-one communication, uniform, statically known messages etc.), this implementation could be considered overhead and in a sense it certainly is. On the other hand, the implementation itself becomes rather simple: Currently it consists of 96 lines of Modelica- and 216 lines of C-Code. In case a faster solution is needed, modbus should be trivial to port to whatever IPC-mechanism seems appropriate. Additionally, this strategy makes it unnecessary to store and continuously calculate the 3d geometry. Depending on the scene, this might yield significant runtime and

¹It has been ported to windows, too.

memory improvements over methods like the Multi-Body visualization.

To further reduce the size of the interface (and make it more convenient to implement), all methods provided by the Modelica3D API accept *named* parameters and allow to emit some of them (using defaults instead). Internally modbus implements this by storing the parameters in a dbus map-object. This yields some further overhead (dbus' internal type-checking becomes quite useless), but allows more selective updates on graphical objects (e.g. it would be possible to only change the Z-axis location of an object without even knowing its X- and Y-axis locations).

Method name	Description
loadSceneFromFile	Loads a complete scene from a file
createMaterial	Create a material primitive
applyMaterial	Use a material on an object
createBox	Create a box primitive
createBoxAt	Create a box primitive, with a given orientation
createSphere	Create a sphere primitive
createCylinder	Create a cylinder primitive
createCylinderAt	Create a cylinder primitive, with a given orientation
createCone	Create a cone primitive
createConeAt	Create a cone primitive, with a given orientation

Table 1: Modelica3D setup-methods

3.3 Alternatives

As already mentioned, the choice of dbus for message exchange was mainly due to pragmatic reasons. Any other platform-independent IPC solution might suffice as well. Albeit, there is a fundamentally different design that needs some discussion. In certain settings, *every* message exchange, no matter how lightweight, may cause a too big delay:

Consider a real-time system running at 60 or more fps and visualizing large sets of objects (e.g. a scene in a game engine). Since synchronous message exchange requires at least 2 context switches, and a context switch is rather costly [6], we can estimate a theoretical upper limit of 10^5 simultaneously animated objects. Any practical limit will of course be much smaller, since not only context switches are required. Basically, this means, that, independent of the

visualization or simulation complexity, a system composed of some thousands of objects that shall be visualized, cannot be rendered in real-time, when message-passing is used.

So instead of sending lots of small messages about the state of each object, front- and back-end could use *shared memory* to exchange large chunks of data very fast. Unfortunately, such a solution is hardly platform independent and more complicated to implement (since both sides would need to synchronize their access on that data). But since it is obviously a useful design alternative, further research seems to be appropriate.

3.4 Data structures and operations

Modelica3D comes only with a very small set of data structures. Next to the already mentioned modbus objects, it provides a system state record, a controller model and a definition of object-ids. The system state basically only combines a modbus context object with a connection and a counter for the current frame. The controller in turn wraps the state and provides a sampled boolean signal depending on a selected framerate. It also modifies the state's frame counter according to the current time and can send a stop-message to the back-end at the end of simulation time.

Method name	Description
rotate	Change an object's orientation
moveTo	Change an object's location
moveZ	Move along the Z-axis only
scale	Change the size of an object
scaleZ	Scale along the Z-axis only
setAmbientColor	Sets the ambient color value of a material
setDiffuseColor	Sets the diffuse color value of a material
setSpecularColor	Sets the specular color value of a material
setMatProperty	Changes a given (named) material property

Table 2: Modelica3D modification-methods

Id-objects are currently only heap-allocated strings. But on demand, they might be easily exchanged with a more complex internal implementation (e.g. if the library would want to implement hashing or collect statistics on the objects).

Most methods in the Modelica3D API fall into two distinct groups: There are operations that describe the *setup* of a scene (table 1) and operations that *modify* a scene dynamically (table 2). The difference between them is that the latter ones need a frame number, which works as a logical clock that describes, *when* such a modification takes effect, while the former ones are always interpreted once at the beginning of the animation. The only exception from that pattern is the stop-operation. Sending this message tells the client to stop listening for further messages.

The set of currently implemented operations is rather small. But due to the design of Modelica3D, additional operations might be added by simply extending the package (and at least one backend). No recompilation of the C-library is required.

Listing 1: moveTo-method in Modelica

```
function moveTo
  input State state;
  input Id id;
  input Real p[3];
  input Integer frame=state.frame;
  output String r;

  protected
  Message msg = Message(TARGET,
    OBJECT, INTERFACE, "move_to");
  algorithm
    addString(msg, "reference",
      getString(id));
    addReal(msg, "x", p[1]);
    addReal(msg, "y", p[2]);
    addReal(msg, "z", p[3]);
    addInteger(msg, "frame", frame);
    r := sendMessage(state.conn, msg);
end moveTo;
```

Implementing an operation in Modelica is not difficult. Listing 1 shows the moveTo function is implemented. It consists of allocating a message object (from the dbus-connection constants for the target and the dbus-interface and the method's name), adding parameters to that message, and finally sending it. Further operations should follow that pattern.

3.5 Back-ends

Currently, Modelica3D contains two back-end implementations. They demonstrate two distinct kinds of visualization tools. The first tool, blender [11], is a 3D-modeling tool which can render high-quality movies. Blender provides a python interpreter for scripting purposes. Thus it was a natural choice to implement the back-end parts in python.

Listing 2: moveTo-method in blender

```
@mod3D_api(reference = defined_object,
  frame = positive_int)
def move_to(self, reference,
  x=None, y=None, z=None,
  frame=1, immediate=False):
  o = data.objects[reference]
  context.scene.frame_set(frame=frame)
  if immediate:
    o.keyframe_insert('location',
      frame=frame - 1)

  if (x != None):
    o.location.x = x
  if (y != None):
    o.location.y = y
  if (z != None):
    o.location.z = z

  o.keyframe_insert('location',
    frame=frame)

  return reference
```

Listing 2 shows the implementation of the moveTo-method in the blender back-end. The mod3D_api-decorator is responsible for lifting a python function into a dbus-method. That lifting is (due to the uniform signature) the same for all back-end methods. Additionally certain runtime checks might be added (e.g. checking if a given object-reference actually exists, a number is positive etc.).

Since blender provides access to its internal data representation (data.objects), the rest of the method is straight-forward. It directly changes the object's coordinates (if provided by the client) and inserts an animation key-frame (allowing for interpolated movement, if necessary).

The other back-end was implemented using OpenSceneGraph [9], a free 3D-engine. Unlike blender, it does not provide modeling facilities. Instead, its scope is fast, real-time rendering. That way we demonstrate how Modelica3D might also be used in interactive applications².

4 Usage

In this section we show, how Modelica3D can be used to visualize different kinds of simulations. First, we will show how Modelica3D can handle state-of-the-art visualizations on the basis of the MultiBody library. Second we will describe the visualization of a simple, structurally dynamic system.

²The graphical output, the input needs to be handled with some other tool or library.

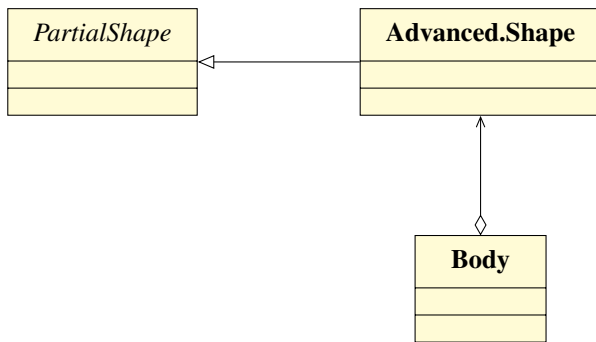


Figure 3: MultiBody visualization-class structure

4.1 Visualizing MultiBody

As mentioned earlier a popular (if not the most popular) method of visualization comes with the Modelica Standard Library: All models from the MultiBody library can be visualized according to their geometrical structure. Since a 3D-mechanical library naturally contains information about the location and relative rotation of objects, visualization is straight-forward.

This makes the MultiBody library a good example of how Modelica3D can be used in such existing complex hierarchies. In this use-case, all visualization information culminates in one class, the `PartialShape` (Figure 3). That class basically consists of the shape parameters (length, material etc.) and a translation matrix. This gives us an insertion point of where to insert the Modelica3D functionality. In a first step, we introduced a controller object into this class, to hold the Modelica3D context information. Since this controller needs to be unique among all shapes, it is naturally marked as **outer** (Listing 3).

 Listing 3: Additional fields of `PartialShape`

```

outer M3D.Controller m3d_control;
Id id;
Id mat;
String res;
discrete Real[3] pos;
modcount.Context initContext
= modcount.Context();
    
```

Additionally an object-id is added for both the shape's material and geometry. The variable `pos` holds the current position (resolved from the translation matrix), while `res` captures the result of each operation (ensuring that they are evaluated at least once). Finally a `modcount`-context object is used to ensure singleton evaluation of message generation. With those fields present, the animation dynamics can easily be implemented by **when**-algorithms:

 Listing 4: `PartialShape` algorithmic dynamics

```

when initial() and
  modcount.get(initContext) <> 1 then
  id := shapeDescrTo3D(m3d_control.state,
    shapeType, length, width, height,
    lengthDirection);
  mat := M3D.createMaterial
    (m3d_control.state);
  M3D.setAmbientColor(m3d_control.state,
    mat, color[1] / 255, color[2] / 255,
    color[3] / 255, 1.0, 0);
  M3D.setSpecularColor(m3d_control.state,
    mat,
    specularCoefficient * color[1] / 255,
    specularCoefficient * color[2] / 255,
    specularCoefficient * color[3] / 255,
    1.0, 0);
  M3D.applyMaterial(m3d_control.state,
    id, mat);
  modcount.set(initContext, 1);
end when;

when m3d_control.send and
  modcount.get(initContext) == 1 then
  pos := r + Frames.resolve1(R, r_shape);
  res := M3D.rotate(m3d_control.state,
    id, R.T, m3d_control.state.frame);
  res := M3D.moveTo(m3d_control.state,
    id, pos, m3d_control.state.frame);
end when;
    
```

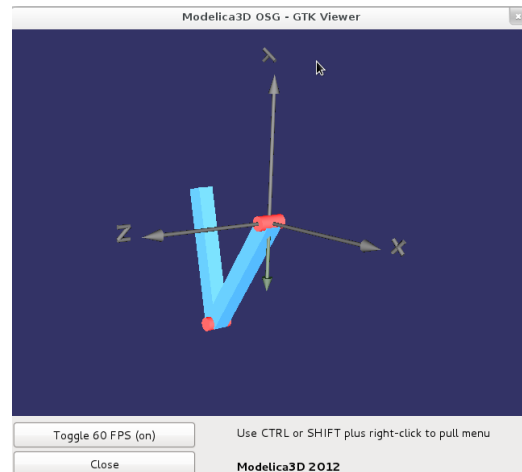


Figure 4: MultiBody visualization with Gtk+/OpenSceneGraph back-end

In this example we omitted some dynamics like changing lengths or colors, since this is unused in our example models (all shapes basically remain constant during simulation). If necessary, those details can be added here easily. Also messages are always sent, even if there is no movement on every frame. A more sophisticated implementation could detect a relevant

change in the model and decide whether or not to update the visualization state.

With this small extension, we were able to simulate and visualize the examples from the standard library with the OpenModelica simulation tools (Figure 4). Thus we successfully demonstrated that by only using standardized techniques, we could visualize complex models.

4.2 Variable-structure modeling

A variable-structure model is a model which can consist of different systems of equations (with different numbers of equations) and different variables depending on the simulation time. This is of interest, when a model has different levels of detail. Another application is to change the model's behavior described by a different set of equations.

Simulation engines like Dymola, SimulationX and OpenModelica do not support such changes. To overcome this drawback and still be able to use common simulation engines for the simulation of a model, a Python framework was introduced in [10]. This framework allows the user to specify a variable-structure model. The user can specify an arbitrary number of models and switches between these models. The user also has to specify how the new model should be initialized with the end values of the old mode.

For now the simulation engines Dymola, OpenModelica and Simulink are integrated in the framework. But the framework is implemented in such a way that other environments can be added quite simply. After specifying the model the Framework starts to simulate the first model in the chosen simulation environment. The model needs a stop condition which specifies when another model should be used and defines the next model. The framework uses this information to switch to the next model and initialize this model with the correct values.

We demonstrate this approach with a simple bouncing ball model. This model could of course be modeled without the variable-structure approach, but it is used for didactic purposes for the modes are easy to understand and the results are good to visualize.

This model consists of two separate modes. The first is the common falling mass model which is valid as long as the ball does not touch the ground. As soon as it touches the ground the ball is modeled as a spring/damper system and therefore the elastic deformation of the model and the bouncing back off the ground can be modeled easily. As soon as the ball leaves the ground again the falling mass model is used

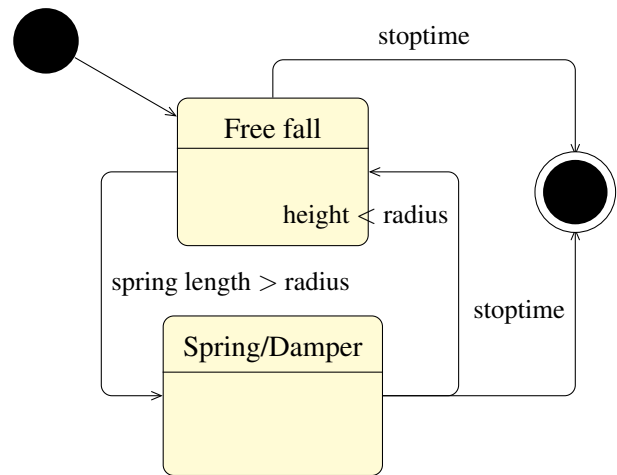


Figure 5: Statechart of the bouncing ball variable-structure model

again. Figure 5 shows a statechart with the two modes and the switching condition is presented.

Simulating this model with the framework and plotting the center of the ball results in the plot shown in figure 6. Here it can be seen, that the center point of the ball reaches below the radius (1.0) of the ball. This effect is caused by the elasticity of the ball in the spring/damper mode.

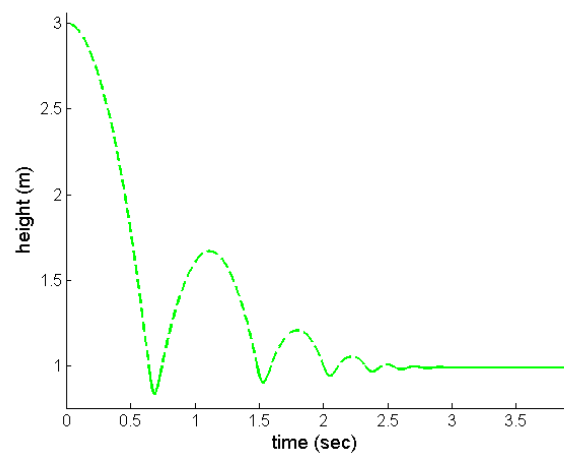


Figure 6: Center point of the bouncing ball model

A simulation of a variable-structure model with the Python framework starts simulations of the different modes sequentially. To be able to visualize such simulation results using Modelica3D, the models describing the states of the system need to fulfill two requirements:

First, they need to work on a *common* scene. Setting up such a scene is trivial: Either by directly loading it into the rendering tool at start or by creating a

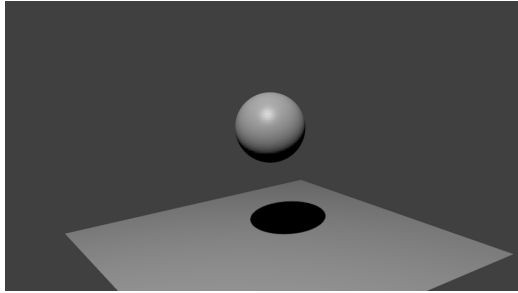


Figure 7: Ball in free-fall mode

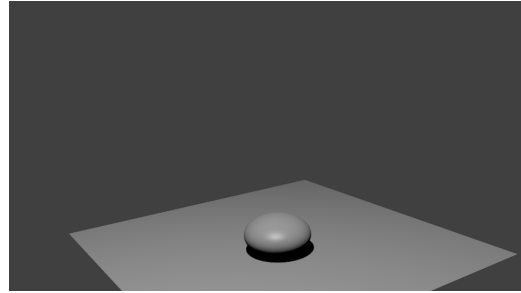


Figure 8: Ball compressed

dedicated initial state that handles all setup commands from the Modelica3D API. Here the decision of using IPC instead of direct linking pays off: Since the rendering tool runs only once, no special treatment for structurally dynamic systems is necessary. Second, all models need to know which parts of the scene they modify. In our example, both models need to know the name of the ball. This *visualization interface* cannot be statically checked.

Listing 5: Free-fall visualization

```
algorithm
  when initial() and
    modcount.get(initContext) <> 1 then
    ball := M3D.objectId("Ball");
    modcount.set(initContext, 1);
  end when;
  when m3d_control.send and
    modcount.get(initContext) == 1 then
    M3D.moveZ(m3d_control.state,
      ball, h, m3d_control.state.frame);
  end when;
```

In our example, we took a simple approach to modeling: The scene consists only of a plane representing the ground and a sphere for the ball. Camera and some lighting is added by blender. In free-fall mode, the only thing to change is the location of the sphere on the Z-axis (Listing 5). On-ground, we model the compression of the ball by scaling and moving the sphere along the Z-axis.

Since the python framework controls the activation and deactivation of the states (by extending the physical models with terminal-conditions etc.), this approach works seamlessly: Figure 7 shows the ball falling towards the plane. The compression is captured in figure 8. Naturally, the true visual effect of bouncing can not be shown in single images, but only when viewing the whole animation.

5 Evaluation

We evaluated a development version of Modelica3D (enhanced with the ability to group objects on the back-end for simpler handling of complex scenes) in a case study of a solar-thermal hydraulic system, which is integrated in the structure of a building envelope. For this objective, several sub-steps had to be realized.

5.1 Modelica3D extensions of the physical models

First, the component models of the library BuildingSystems³ were extended with the ability to have a representation within a 3D scene and to show values such as temperatures, pressures or mass flow rates. Figure 9 shows this extension procedure for the example of a 1D-segmented thermal hydraulic model of a tube. The new model class PipeStraightVis3D was derived from the existing physical model class PipeStraight and from a general model class for 3D representation ModelVis3D.

The model extension comprises the definition of the shape of the 3D sub-primitives (here the cylinder pieces of the segmented fluid volume), the combination of them in a common container, the definition of the material (the appearance in the 3D scene) incl. the link to the sub-primitives, the alignment and merging of the sub-primitives to the common 3D representation and the mapping of the physical values to a graphical representation within the 3D model (in this case the fluid temperature of each fluid segment).

On a next level, several 3D-extended tube models and a 3D-extended pump model were combined to a simple thermal hydraulic loop. Figure 10 shows the 2D diagram of the Modelica system model on the left and the corresponding 3D animated scene on the right.

³<http://www.modelica-buildingsystems.de>

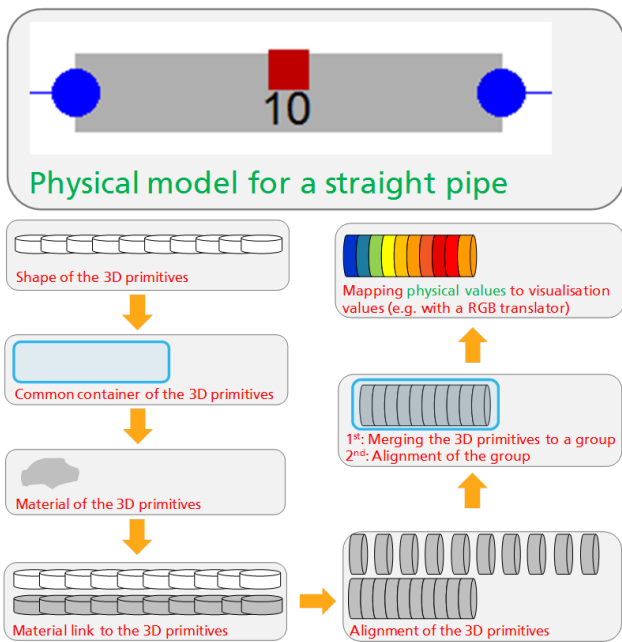


Figure 9: Extending a physical model for the use in Modelica3D

5.2 Case study of a solar thermal system

As the first complex application of the 3D visualization method, a solar thermal system for warm water production was used (Figure 10 left). The components of the solar thermal system are two evacuated tube collectors with a total aperture area of 6.34 m² and a hot water storage with a volume of 400 liters. The roof collector is aligned to the south and tilted with an angle of 30°. An external plate heat exchanger transfers the produced thermal energy from the solar loop to the storage loop. With the help of a two-point-controller the solar pump and the storage pump are switched on,

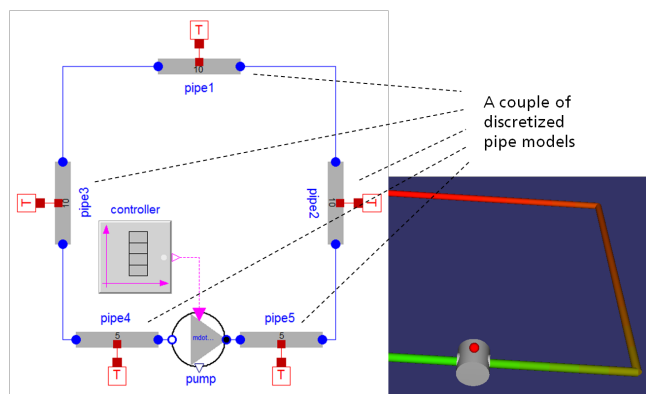


Figure 10: 2D and 3D representation of a thermal hydraulic loop

if the collector outlet temperature is 4K higher than the temperature in the lower part of the storage. As climate boundary conditions weather data from Hamburg (Germany) were used.

In the simulation scenario a load process for the thermal water storage over a time period of 24 h during a summer day were calculated. At the beginning of the load process the fluid temperatures in the collector, in the pipes and in the storage was set to 20 °C. Figure 10 (right) shows the described solar thermal system as a graphical 2D diagram, based on the "3D-extended"-components of the BuildingSystem-library.

Figure 11 illustrates the simulated transient load process for the summer day, described by the most important system variables such as the solar irradiation on the collector, the mass flow rate of the storage pump, the collector outlet temperature and the storage temperature at the bottom.

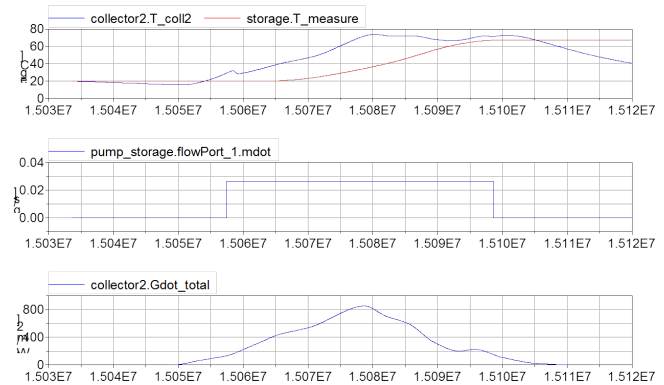


Figure 11: Simulated load process of the solar thermal system

For a clear representation within a 3D scene, the model of the solar thermal system was embedded in the 3D model of a building envelope. The 3D building envelope was modeled as a pure geometrical representation without any physical behavior. In this manner, realistic geometries and positions of different technical components of the solar thermal system (tube lengths and diameters, the required space of the storage and the collectors etc.) can be visualized. Figure 12 shows a snapshot of a the visualized transient load process of the storage during the hours before noon during a summer day in Hamburg. The different colors illustrate the temperatures of the fluid within the collector model, the tubes and the warm water storage from cold (blue) to warm (green). Because the collectors are serial connected and the cold fluid enters at first the left collector, the temperature gradient within the segmented collector model increases from left two right.

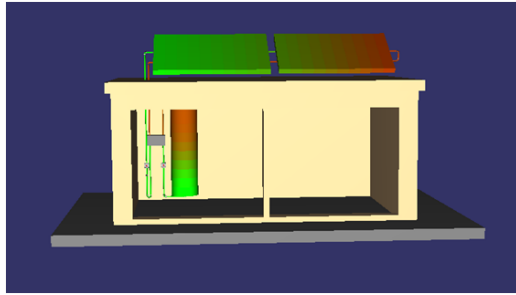


Figure 12: 3D-scene of the solar thermal system

5.3 Results

The result of the evaluation can be summarized as follows:

- The developed method allows a performant representation of 3D scenes with a large quantity of animated graphical 3D elements.
- It is possible to represent complex 3D scenes with the unchanged Modelica code in different 3D environments (eg. Blender and OpenSceneGraph)
- A 3D modeling editor for a time efficient and correct configuration of complex 3D Modelica scenes is absolutely necessary

6 Conclusion

Modelica can be extended too support 3D-visualization of experiments. That extension can completely be implemented in form of a library by only using already standardized techniques. By choosing a loosely coupled, distributed architecture, the extension can support different back-ends and itself be extended easily. Additionally, innovative use-cases as variable-structure modeling are supported by this approach.

6.1 Obtaining Modelica3D

A public version of Modelica3D can be published under the terms of the GNU General Public License. The project page can be found at <https://mlcontrol.uebb.tu-berlin.de/redmine/projects/modelica3d-public>.

References

- [1] Modelica - a unified object-oriented language for physical systems modeling, 2010.

- [2] T. Bellmann. Interactive Simulations and Advanced Visualization with Modelica. In *Proceedings of the 7th Modelica Conference, Como, Italy*, 2009.
- [3] V. Engelson. 3D Graphics and Modelica-an integrated approach. *Linköping Electronic Articles in Computer and Information Science. Linköping universitet*, 2000.
- [4] Peter Fritzson, Peter Aronsson, Håkan 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*, pages 83–90, 2005.
- [5] Thomas Hoeft and Christoph Nytsch-Geusen. Design and validation of an annotation-concept for the representation of 3d-geometries in modelica. In *Proceedings of the 6th International Modelica Conference*, 2008.
- [6] C. Li, C. Ding, and K. Shen. Quantifying The Cost of Context Switch. In *Proceedings of the 2007 workshop on Experimental computer science*, page 2. ACM, 2007.
- [7] R. Love. Get on the D-BUS. *Linux Journal*, 2005(130):3, 2005.
- [8] H. Lundvall, P. Fritzson, and B. Bachmann. Event handling in the openmodelica compiler and runtime system. Technical report, Technical Report 2, Dept. Computer and Information Science, Linköping Univ, 2008.
- [9] Paul Martz. *OpenSceneGraph Quick Start Guide*, 2007.
- [10] A. Mehlhase. A Python Package for Simulating Variable-Structure Models with Dymola. submitted, feb 2012.
- [11] Ton Roosendaal and Stefano Selleri. *The Official Blender 2.3 Guide: Free 3D Creation Suite for Modeling, Animation, and Rendering*. No Starch Press, June 2004.

Proposal for a Standard Time Series File Format in HDF5

A. Pfeiffer¹, I. Bausch-Gall², M. Otter¹

¹DLR Institute of System Dynamics and Control, Oberpfaffenhofen, Germany

²BAUSCH-GALL GmbH, Munich, Germany

Andreas.Pfeiffer@dlr.de, Ingrid.Bausch-Gall@bausch-gall.de, Martin.Otter@dlr.de

Abstract

This paper describes a proposal for a standard to store the results of dynamic systems simulations in form of time series data persistently on file. The reasons to develop such a standard are explained, as well as the decision to use the HDF5 file format as a basis. The meta-information to be stored on file is mainly deduced from the Functional Mockup Interface standard. Two variants are analyzed: Storing the meta-data either with a set of tables or in a hierarchy. Usability and performance measurements are utilized for the selection.

Keywords: Simulation Results; File Format; Time Series; Standard; HDF5; MTSF, FMI

1 Introduction

Many simulation programs store their simulation results in an own specific file format. However, modelers have to utilize simulation results from different tools in different ways, e.g. plotting in company specific formats, comparing the data with results from another simulation program or computing FFTs (Fast Fourier Transforms). Since often one tool is not suited for all these tasks, users or tool vendors have to implement API functions to access the result data from other programs. This is time consuming and has to be adapted when the format changes. Every program stores different information. Some store only the results, other more information such as units and names of signals. Many programs provide an open export of ASCII or CSV files, which makes data access easy. However, information supplied in these formats is not complete, reading the files is inefficient and storing and retrieving large amounts of data is not practical.

These issues exist since decades for almost all simulators in many physical domains. Many simulators offer a more or less mighty environment for result evaluation. But this is not their main development goal. Scripting tools such as Matlab [M12], Scilab [TSC12] or Python [P12a] are better suited to automate plotting of results with fine control of the layout, to generate standardized result evaluation re-

ports, to perform signal processing (e.g. FFT), to compare with measurements, to run Monte Carlo simulations, or to perform optimizations over many simulations etc. The basic problem is then how to connect a simulation with a scripting environment. With a standardized time series file format, the approach from Figure 1 simplifies the task a lot, since simulation environments could generate files in this format and scripting tools could read files in this format directly.

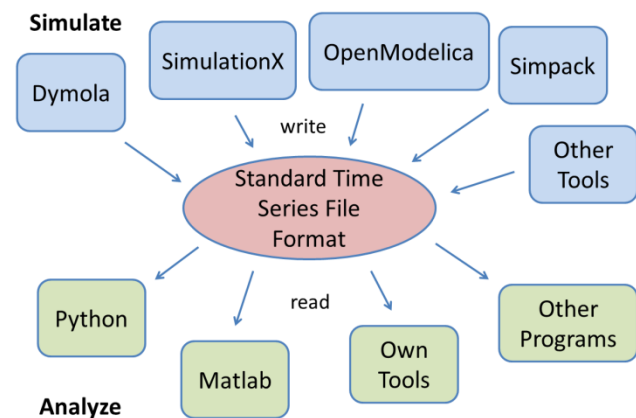


Figure 1: Standard time series file format and its interaction with tools.

In 2010, version 1.0 of the FMI (Functional Mockup Interface) standard was developed for the low level exchange of models and for co-simulation [MC10]. More than 30 tools support this standard already. Further progress can be achieved if these programs would support, at least optionally, the same result file format. For example, this would make it practical to automatically compare results of the same FMI model in different environments, and therefore the FMI import and export between tools could be tested in a much better way.

A standardized file format for simulation results would also be helpful for the Modelica community: More and more different Modelica simulators come to the market. Many components, models and libraries are developed in Modelica. They might be used in different simulators. It is necessary to compare results computed by different simulators automatically. In particular, the Modelica Association plans

to supply reference results for all simulation models available in the Modelica Standard Library [MA10]. This is only practical, if the Modelica tool vendors agree on a standardized result file format. Apart from testing, it might be desirable to collect results e.g. of all slaves in a co-simulation environment in one file.

1.1 Time Series Data

The basic purpose of the proposed file format is the efficient and compact storage of time series data, as shown in Table 1: The first column contains the values of the independent variable, usually time (but might be also another quantity, e.g. frequency), whose values must be monotonically increasing. A discontinuity occurs, if a value appears several times (here: at 0.4). Variable v is an example of a variable that depends on time t .

Table 1: Example for time series data.

Time t	Variable v	Variable w	...
0.0	2.8		
0.2	3.2		
0.4	5.1		
0.4	7.2		
0.5	6.9		
0.6	5.5		

If the variable is a continuous-time variable, then $v = v(t)$ is a continuous function and there exist also values of v between the tabulated points. Such intermediate points can be computed by interpolation of the tabulated values. If the variable is a discrete-time variable, then v is computed by a sampled data system at the values of the provided time instants. A value between the time points is not defined for v . If necessary, $v(t)$ with $t_j < t < t_{j+1}$ can be associated with the previous value $v(t_j)$ (= hold-semantics).

1.2 Name of the Standard

Results from the numerical integration of time dependent differential algebraic equations with discrete variable changes are typical *time series*. Since the standard shall be discussed, finalized and released by the *Modelica Association*, it is proposed to call it “Modelica Association Time Series File Format”, shortly *MTSF*. This name is also used as the current extension of the corresponding files (e.g. robot.mtsf).

2 Selection of Basic Data Format

As a first step we collected requirements for such a file format and evaluated several existing formats against these requirements [BP11].

2.1 Requirements for the Result Format

A format for a time series file should fulfill the following requirements:

- Small and huge amounts of data (more than 10 GBytes) must be written fast and efficiently.
- Extraction of data from small and huge files must be fast.
- The format must be an internationally accepted standard.
- The standard has to be open.
- The format has to be also accepted by simulator developers outside of the Modelica community.
- It has to be future proof, which means stable support by the developers of the standard is expected and it has to be supported by many tools.
- The format should handle at least all data types of the FMI standard 1.0 [MC10] and the coming FMI standard 2.0 [MC12].
- It should be possible to add more data, if desired (e.g. diagrams of the model).
- APIs to standard programming languages like C, C++ and Fortran should exist.
- It should be easily accessible from scripting programs such as Matlab, Python, and others.

2.2 HDF5 Format

HDF, HDF4 and HDF5 (Hierarchical Data Format) [THG12a] are a set of file formats and libraries designed to store and organize large amounts of numerical data, originally developed at the NCSA (National Center for Supercomputing Applications at the University of Illinois). In 2005, the Hierarchical Data Format group was spinning off from NCSA as a non-profit corporation to ensure continued development of HDF technologies and the continued accessibility of data currently stored in HDF [NCS+12]. The HDF format, libraries and associated tools are available under a liberal BSD-like license. HDF is supported by many commercial and non-commercial software platforms, including Java, Matlab, IDL and Python.

The freely available HDF distribution consists of an API to access HDF files (implemented in C, with layers for C++, Fortran and Java), command line utilities, test suite sources, and the Java-based HDF Viewer to directly inspect HDF files. The currently existing two versions HDF4 and HDF5 differ signif-

icantly in design and API. The newer, more powerful HDF5 format consists of a hierarchy of objects where the leaf objects are arrays. The dimensions of an array need not be known in advance and may be even constructed incrementally (as it naturally occurs in simulations). Many native data types are supported including all C data types. Furthermore, data can be compressed and graphics as well as videos can be stored. On the HDF web page applications with terabyte file sizes are reported (http://www.hdfgroup.org/why_hdf).

In [P10] a good overview of the features of HDF5 is given. It is suggested to use HDF5 to store simulation data. The reference highlights the following features of HDF5: The tree structure for convenient storage of data; HDF is a numerical aware middleware; the files and APIs allow portability, maintainability, compatibility of the user software; the openness of the software and the trustworthiness of the support.

2.3 Alternatives to HDF5

In order to handle efficiently large result data, only binary formats seem to be suitable. In principal also zipped xml-files might be applicable, but there seems to be still quite a large overhead to store and retrieve structured numerical data in such a format.

There exist also other open source binary file formats, in particular:

- **NETCDF**¹ from UCAR (University Corporation for Atmospheric Research). The latest version of NETCDF is a subset of HDF5 and the NETCDF files are therefore compatible to HDF5 (see “Format Descriptions” in <http://en.wikipedia.org/wiki/NetCDF>).
- **CDF**² from NASA. The CDF format is not compatible to HDF5. CDF seems to be also widely used and is, e.g. supported in Matlab and Python. CDF supports a set of arrays, but it does not support an object hierarchy. In this respect the HDF5 format is more powerful.

Another alternative could be to not base the design on a general purpose file format, but on a special binary format dedicated solely to time series data:

- Such a format could be newly designed and implemented. However, it would be a large effort to develop, implement and support an API that

writes time series data in a subset of a HDF5-like data structure. Therefore we decided to not follow this approach.

- Another option would be to use one of the formats of ASAM (Association for Standardisation of Automation and Measuring Systems) [A12]. ASAM was founded in 1998 as an initiative of German car manufacturers with the goal of offering a platform for the development of universal standards such as MCD-2 MC, MDF, HIL V1.0.1 and ODS. A standard like ASAM MDF (Measurement Data Format) can be compared to the MTSF approach. It is designed to store and retrieve data from measurements. This standard is widely used in automotive industry. HDF5 and ASAM standards are, e.g., compared in [PA11]. There exists no open source API from ASAM to read and write data. The standard texts are available for ASAM members (with expensive membership fees for industrial partners) or can be bought for a pricey fee. For these reasons, ASAM standards seem to be not suited as general exchange format for time series result data between many tools.

Since all requirements of section 2.1 are fulfilled by the HDF5 format and there seems to be no equally suitable competitor, we decided to base the MTSF format on HDF5. Once the base file format is decided, the important question is what data to store? Our main target is to store simulation result data from tools that support the FMI standard [MC10, MC12]. Therefore, the time series data and associated meta-information to be stored is based on this standard.

3 Structure of the File Format

The basic structure of an MTSF file is shown by means of an example using screen shots from HDFView [THG12b]. The example file is based on the numerical integration of a Functional Mockup Unit (FMU) [MC10] by the open source simulator PySimulator [PHH+12]. The FMU was generated by Dymola [DS12] from the model `Modelica.Mechanics.Multibody.Examples.Systems.RobotR3.fullRobot` of the Modelica Standard Library [MA10]. The complete hierarchy of the result file is shown in Figure 2.

On the top level, the file shows two groups named *ModelDescription* and *Results*. *ModelDescription* contains the meta-information of the variables. The time series data of these variables is stored under *Results*. In order to read the result data of one or more variables, parts of the *ModelDescription* in-

¹ <http://www.unidata.ucar.edu/software/netcdf>

² <http://cdf.gsfc.nasa.gov>

formation has to be inquired in order to determine the location where the result data is stored.

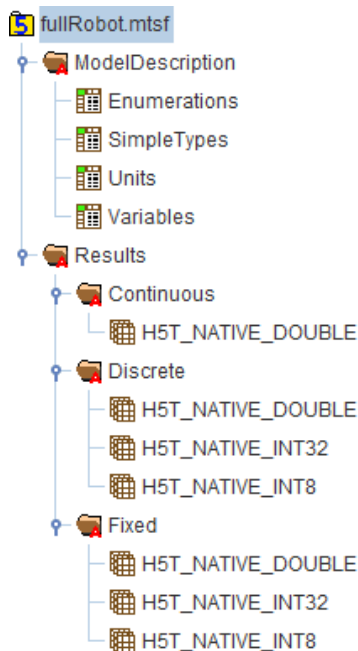


Figure 2: HDF5 hierarchy of the example result file.

The root directory has an attribute *mtsVersion* that contains a string value for the version of the underlying MTSF format, see Figure 3. All groups and datasets from Figure 2 are described in the next subsections.

Name	Value
mtsVersion	0.3

Figure 3: HDF5 attribute on root level of the result file.

3.1 Model Description

The HDF5 group *ModelDescription* contains a set of attributes (see Figure 4) which give (optional) information about the source of the model used for the experiment. The information is based on the coming FMI 2.0 definition [MC12].

Name	Value
author	
description	6 degree of freedom robot with path planning, controll
generationDateAndTime	2011-03-30T07:52:02Z
generationTool	Dymola Version 7.4 FD01, 2011-02-24
modelName	Modelica.Mechanics.MultiBody.Examples.Systems.Rc
variableNamingConvention	structured
version	3.2

Figure 4: HDF5 attributes of group *ModelDescription* in the example result file.

Variables

The HDF5 dataset *ModelDescription/Variables* (see Figure 6) defines the variables whose data is stored in the file. The HDF5 type definitions of the dataset *Variables* are displayed in Figure 5:

- *name* contains the names of the respective variables.
- *simpleTypeRow* defines the data type and the unit of the variable by providing the row index of the related simple type in dataset *ModelDescription/SimpleTypes* (see below). For example *simpleTypeRow* = 33 means that the type is defined in row 33 of *SimpleTypes* which means *Modelica.Slunits.Angle* (see Figure 8).
- *causality* and *variability* are HDF5 enumerations and provide information about the nature of the variable.
- *description* is a short description string of the variable.
- *objectId* and *column* provide the information where the data is stored for this variable (more details are given in section 3.2).
- *negated* is introduced to enable negated alias variables. It can only have the values 0 for *false* or 1 for *true*. The value 1 indicates that the values for this variable (stored in the data matrices under *Results*) have to be negated.

Name	Type
name	String, length = 56
simpleTypeRow	32-bit unsigned integer
causality	enum (input=2 local=4 option=5 output=3 parameter=1)
variability	enum (constant=1 continuous=5 discrete=4 fixed=2 tunable=3)
description	String, length = 144
objectId	Object reference
column	32-bit unsigned integer
negated	enum (false=0 true=1)

Figure 5: HDF5 variable types for the columns of the dataset *ModelDescription/Variables* in the example result file.

Instead of *objectId* and *column* it would also be possible to use an HDF5 region reference. This is a HDF5 link to the region of a data matrix, in our cases, e.g. a column of one of the matrices under *Results/Continuous*. A typical region reference looks like 0:3396963 { (0,685)-(599,685) } in HDFView where 3396963 is the HDF5 object id of the matrix. The region is selected by row 0 up to row 599 of column 685. The drawback of the region reference is that it is not supported to link to a whole column of a matrix. The row indices of the region have to be specified, too. Because the number of result points is generally not known before a simulation, the row indices of the region reference have to be updated at the end of the simulation process, which is quite an overhead if many variables are present.

	name	simpleTypeRow	causality	variability	description	objectId	column	negated
200	axis1.motor.D	0	1	2	Damping constant...	2831	2180	0
201	axis1.motor.J	43	1	2	Moment of inertia ...	2831	2179	0
202	axis1.motor.Jmotor.J	43	1	2	Moment of inertia	2831	2050	0
203	axis1.motor.Jmotor.a	35	4	5	Absolute angular ...	4013	153	0
204	axis1.motor.Jmotor.flange_a.phi	33	4	5	Absolute rotation ...	4013	325	0
205	axis1.motor.Jmotor.flange_a.tau	53	4	5	Cut torque in the fl...	4013	93	1
206	axis1.motor.Jmotor.flange_b.phi	33	4	5	Absolute rotation ...	4013	325	0
207	axis1.motor.Jmotor.flange_b.tau	53	4	5	Cut torque in the fl...	4013	503	1
208	axis1.motor.Jmotor.phi	33	4	5	Absolute rotation ...	4013	325	0

Figure 6: Dataset *Variables* in group *ModelDescription* of the example result file.

Simple Types

Name	Type
name	String, length = 54
dataType	enum (Boolean=3 Enumeration=5 Integer=2 Real=1 String=4)
quantity	String, length = 28
relativeQuantity	enum (false=0 true=1)
description	String, length = 1
unitOrEnumerationRow	32-bit integer

Figure 7: HDF5 variable types for the columns of the dataset *ModelDescription/SimpleTypes* in the example result file.

The dataset *SimpleTypes* (see Figure 8) contains a definition of the simple data types used in the variable description. The HDF5 data type definition of the columns is depicted in Figure 7. The simple data type can optionally have values for the string fields *name* and *quantity*. *dataType* is an HDF5 enumeration that specifies the basic data type (for example *Real* has the value 1). The default value for *unitOrEnumerationRow* is -1 (means no row) and for *relativeQuantity* it is 0. The *relativeQuantity* can only have values of 0 or 1 that represent *false* or *true* (this is only relevant if unit conversion takes place) If *dataType* is equal to 5 (= Enumeration) the value of *unitOrEnumerationRow* corresponds to a row in the dataset *ModelDescription/Enumerations*, otherwise to a row in the dataset *ModelDescription/Units*.

Units

Each simple data type can have a unit and several display units. Display units for one simple data type can be defined by using a row block in the dataset *Units*, see Figure 9 and Figure 10. A unit can have three different modes: base unit, display unit or default display unit. The value of *unitOrEnumerationRow* has to correspond to a row in *Units* with mode = 0 (BaseUnit), if there is a unit definition. If some display units apply for this base unit, they have to be listed in the rows below the base unit. Each mode of the display units can be 1 (DisplayUnit) or 2 (DefaultDisplayUnit). Only one display unit may have *mode* = 2. If no display unit has *mode* = 2, the base unit is used as default display unit. The base unit can only have *mode* = 0. If there is no unit (and no enumeration) for a simple data type, then the value for its *unitOrEnumerationRow* is equal to -1 (default value).

For example, the simple data type *Time* (see row 56 in Figure 8) is a Real data type with a unit that is defined in column 33 of the dataset *Units*. Here the base unit is *s* and the display units are defined by rows 34 up to 37 in the dataset *Units* (Figure 9). So, the display units are: *ms*, *min*, *h* and *d* with corre-

	name	dataType	quantity	relativeQuantity	description	unitOrEnumerationRow
28	Modelica.Mechanics.MultiBody...	5		0		0
29	Modelica.Mechanics.MultiBody...	1		0		-1
30	Modelica.Mechanics.MultiBody...	4		0		-1
31	Modelica.Mechanics.MultiBody...	1		0		-1
32	Modelica.Slunits.Acceleration	1	Accelerati...	0		24
33	Modelica.Slunits.Angle	1	Angle	0		28
34	Modelica.Slunits.Angle	1	Angle	0		30
35	Modelica.Slunits.AngularAccel...	1	AngularAc...	0		27
36	Modelica.Slunits.AngularVelocity	1	AngularVe...	0		26
37	Modelica.Slunits.Capacitance	1	Capacitan...	0		4
38	Modelica.Slunits.Current	1	ElectricCu...	0		2
39	Modelica.Slunits.Diameter	1	Length	0		22
40	Modelica.Slunits.Distance	1	Length	0		22
56	Time	1	Time	0		33

Figure 8: HDF5 dataset *ModelDescription/SimpleTypes* in the example result file.

sponding values for *factor* and *offset* in the style of FMI 2.0 [MC12]. The default display unit is *s*. This allows, e.g. a plotting program to display the results in different units by using the conversion factors stored in the *Units* group.

	name	factor	offset	mode
26	rad/s	1.0	0.0	0
27	rad/s2	1.0	0.0	0
28	rad	1.0	0.0	0
29	deg	57.295779...	0.0	2
30	rad	1.0	0.0	0
31	deg	57.295779...	0.0	1
32	s	1.0	0.0	0
33	s	1.0	0.0	0
34	ms	0.0010	0.0	1
35	min	60.0	0.0	1
36	h	3600.0	0.0	1
37	d	86400.0	0.0	1

Figure 9: HDF5 dataset *ModelDescription/Units* in the example result file.

Name	Type
name	String, length = 9
factor	64-bit floating-point
offset	64-bit floating-point
mode	enum (BaseUnit=0 DefaultDisplayUnit=2 DisplayUnit=1)

Figure 10: HDF5 variable types for the columns of the dataset *ModelDescription/Units* in the example result file.

Enumerations

The dataset *ModelDescription/Enumerations* (see and Figure 12) lists all enumerations that are defined in the model variables, i.e. variables of type Integer that can have only a small number of Integer values and a string is associated with every value. A plot program may then use the enumeration name instead of an integer to mark the value in an axis. The value of *unitOrEnumerationRow* corresponds to a row in the dataset *ModelDescription/Enumerations*, if the data type of a simple type is equal to 5 (= Enumeration). Enumerations do not have units, so there is no conflict with unit definitions.

Name	Type
name	String, length = 14
value	32-bit integer
description	String, length = 81
firstEntry	enum (false=0 true=1)

Figure 11: HDF5 variable types for the columns of the dataset *ModelDescription/Enumerations* in the example result file.

The row of *Enumerations* that corresponds to *unitOrEnumerationRow* has to have *firstEntry* = 1. The *firstEntry* column marks a new row block of enumerations. Each enumeration has a name and an integer value and may have a separate description

string for example, the simple type *StateSelect* is an enumeration type with *unitOrEnumerationRow* = 7, it means in row 7 of *Enumerations* the defining enumeration block starts from “never” (1) up to “always” (5). Values for enumeration types are stored as integer.

	name	value	description	firstEntry
0	NoGravity	1	No gravity field	1
1	UniformGravity	2	Uniform gravity field	0
2	PointGravity	3	Point gravity field	0
3	NoInit	1	No initialization (sta...	1
4	SteadyState	2	Steady state initializ...	0
5	InitialState	3	Initialization with init...	0
6	InitialOutput	4	Initialization with init...	0
7	never	1		1
8	avoid	2		0
9	default	3		0
10	prefer	4		0
11	always	5		0

Figure 12: HDF5 dataset *ModelDescription/Enumerations* in the example result file.

3.2 Time Series Results

The numeric data associated with the defined variables is stored under *Results*. The HDF5 attributes of *Results* in Figure 13 include the most important parameters for the simulation experiment. *ResultType* defines the kind of the experiment, here: *Simulation*. The other attributes depend on the value of *ResultType*. For example, a result type *Measurement* has other attributes than a result type *Simulation*, but the attributes are standardized. The values of the attributes are optional with empty strings as default. Standardized attributes are necessary to exchange the attributes between different tools.

Name	Value
ResultType	Simulation
algorithm	BDF (IDA, Dassl like)
author	Mr. X
cpuTime	
description	
generationDateAndTime	Tue, 17 Apr 2012 09:45:49
generationTool	Python
machine	Pluto
relativeTolerance	1.0E-4
startTime	0.0
stopTime	2.0

Figure 13: HDF5 attributes of group *Results* in the example result file.

The experiment may provide several time series under *Results*. Example names for the time series are *Continuous* for continuous-time variables, *Discrete* for discrete-time variables which change their values only at events, and *Fixed* for variables that do not depend on an independent variable (constants and

parameters). Additional groups might correspond to different clocks (e.g. a group for a periodic sample rate of 2 ms and a group for a periodic sample rate of 7 ms).

The group names of the time series can be freely chosen. Every time series (corresponding to a separate HDF5 group) may be associated with an independent variable. Therefore, each time series group has the attributes *independentVariableRow* and *interpolationMethod*. For example the attribute definitions of groups *Continuous* and *Discrete* are shown in Figure 14.

Continuous		Discrete	
Name	Value	Name	Value
<i>independentVariableRow</i>	0	<i>independentVariableRow</i>	1
<i>interpolationMethod</i>	linear	<i>interpolationMethod</i>	constant

Figure 14: HDF5 attributes of the groups *Results/Continuous* and *Results/Discrete* in the example result file.

The value of *independentVariableRow* is the row index in the dataset *ModelDescription/Variables* and defines the variable that is used as independent variable for the relevant data. In our example the independent variable of *Continuous* is variable *Time* that has a row index of 0. The independent variable of *Discrete* is variable *DiscreteTime* that has a row index of 1. For group *Fixed* the index *independentVariableRow* is equal to -1 in order to indicate that the variables are constant and do not depend on an independent variable.

The value of *interpolationMethod* is *linear*, *constant* or *clocked* and indicates how the numeric data values corresponding to the time series have to be interpreted. *Linear* means that piecewise linear interpolation is suggested between the given points. *Constant* means that the value of a variable for a point of time is held constant until the next point of time. *Clocked* means that no interpolation should be applied and only the values at the stored time points should be

shown in a plot. Typically, *linear* is applied for continuous-time variables, *constant* for discrete-time variables that have an explicit value between event points, and *clocked* for sampled variables.

All time series data under a group like *Continuous* are stored in matrices. The column of such a matrix corresponds to one or more model variables and the row corresponds to the values of the independent variable. All elements of a matrix have the same HDF5 data type and the name of this data type is used as name of the matrix. In Figure 2, there are three matrices under *Discrete* of the types *H5T_NATIVE_DOUBLE*, *H5T_NATIVE_INT32*, and *H5T_NATIVE_INT8*. These are HDF5 data types and mean the matrices have a 64 bit floating type, a 32 bit integer type and an 8 bit integer type, respectively. In the latter matrix, the data of Boolean variables is stored as value 0 or 1. A basic Boolean type is not available in HDF5.

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	0	0
2	0	0	-1	0	0	0
3	0	0	-1	0	0	0
4	0	0	-1	0	-1	0
5	0	0	-1	0	-1	0
6	0	0	-1	-1	-1	0
7	0	0	-1	-1	-1	0
8	0	1	-1	-1	-1	0
9	0	1	-1	-1	-1	0

Figure 16: Parts of the dataset *Results/Discrete/H5T_NATIVE_INT32* from the example result file.

Parts of the matrix *Results/Discrete/H5T_NATIVE_INT32* are shown in Figure 16. Each column of the matrix contains the numeric data of one or more discrete integer variables. The time values for the data are stored in a column of the matrix *Results/Discrete/H5T_NATIVE_DOUBLE* (see Figure 15). The column index is given in *column* of *ModelDescription/Variables* for the independent variable *DiscreteTime*. In the example file the column index

	0	1	2	3	4	5	6	7
9	0.004361142295306215	26.4	-10.7...	-15.0	5.39...	5.39...	6.87...	14.39...
10	0.0043611423912718415	26.4	-10.7...	-15.0	5.39...	5.39...	6.87...	14.39...
11	0.004667320075657221	26.4	-10.7...	-15.0	5.39...	5.39...	6.87...	14.39...
12	0.004667320085177664	26.4	-10.7...	-15.0	5.39...	5.39...	6.87...	14.39...
13	0.11742424242424244	0.0	-0.0	-0.0	0.0	0.0	0.0	0.0
14	1.238620759479847	-26.4	10.79...	15.0	-5.3...	-5.3...	-6.8...	-14.3...
15	1.3486863157736644	-26.4	10.79...	15.0	-5.3...	-5.3...	-6.8...	-14.3...
16	1.3497196258220059	-26.4	10.79...	15.0	-5.3...	-5.3...	-6.8...	-14.3...
17	1.3503821984874294	-26.4	10.79...	15.0	-5.3...	-5.3...	-6.8...	-14.3...

Figure 15: Parts of the dataset *Results/Discrete/H5T_NATIVE_DOUBLE* from the example result file.

is 0. So the first column of *Discrete/HST_NATIVE_DOUBLE* represents the time for all discrete variables. All matrices of a time series group have the same number of rows: They are based on the same independent variable values.

4 Performance Tests with Python, Dymola and Matlab

We used Python 2.7 [Py12] to implement a test environment for writing and reading MTSF files. The Python(x,y) distribution (version 2.7.2.1) [P12b] includes the HDF5 interface h5py (version 2.0.1) [H12], which provides high level interface functions in Python for HDF5 files. In a second step, reading MTSF files by Matlab [M12] is tested.

4.1 Hierarchical Variables Concept

In the initial design phase of the MTSF format a different (alternative) format has been investigated than presented in Section 3. In this section we shortly explain this alternative format (called *hierarchical variables concept*), because it seems to be straightforward to save hierarchically structured variables in a HDF5 group hierarchy. However, the performance measurements in Section 4.2 and 4.4 indicate that the table-based approach of section 3 is better.

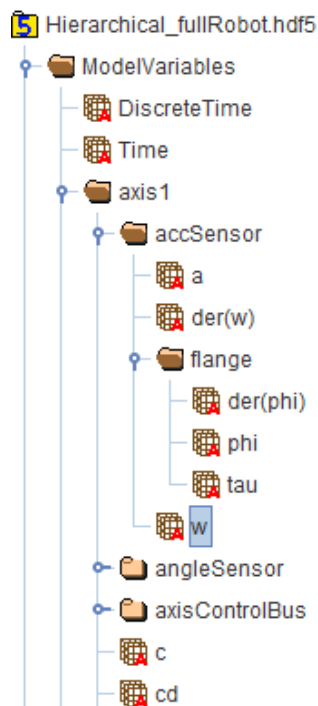


Figure 17: Parts of the HDF5 hierarchy using the hierarchical variables concept for the example result file.

The one to one mapping of hierarchical variable names to HDF5 groups and datasets is the main difference to the MTSF format presented in Section 3. For example, for the variable `axis1.accSensor.w` the dataset *ModelVariables/axis1/accSensor/w* in Figure 17 contains all the necessary information about the variable. The *ModelDescription* group (see Section 3.1) with its compound datasets is not present in this concept.

For first testing purposes the deepest dataset for each variable is only a 1x1 dataset containing an HDF5 object reference to one of the matrices under e.g. *Results/Continuous*. The numeric data for the variable is stored in one of the columns of the referenced matrix. The column index is stored as attribute to the reference dataset. It would also be possible to use an HDF5 region reference instead of the object reference and the column index. The resulting files sizes would only differ slightly.

Additional information (data type, unit, etc.) of each variable can be stored as further attributes. Some attribute examples are listed in Figure 18. To get the whole information included in the MTSF format of Section 3, much more attributes (or dimensions of the dataset) would be necessary. We have not worked it out so far.

Name	Value	Type
columnIndex	685	32-bit integer
dataType	1	enum (Boolean=3 ...
declaredType	Modelica.SIunits.AngularVelocity	String, length = 32
description	Absolute angular velocity of flange	String, length = 35
variability	continuous	String, length = 10

Figure 18: Some HDF5 attributes of each hierarchical variable dataset.

The advantage of the hierarchical variables concept is the hierarchical mapping of the model variable names and the HDF5 groups / datasets in the *ModelVariables* tree. Therefore, HDFView shows automatically a tree of the model variables when browsing through the groups. The main disadvantage of the hierarchical concept is the possibly large number of HDF5 objects building the *ModelVariables* tree. Intuitively, this is similar to a file system: Reading or writing 10 files (corresponds to the MTSF format of Section 3) is more efficient than reading or writing 10000 files (corresponds to the hierarchical variables concept) in which the same data is stored.

In the next subsection we compare the files that result from the two different concepts: hierarchical storage of variable information vs. the final MTSF

format of Section 3 with few HDF5 compound datasets in *ModelDescription*.

4.2 File Sizes

In MTSF files the HDF5 compression of objects with the gzip algorithm can be used which is very effective for the meta-information, whereas compression with gzip in the hierarchical variable concept is technically not possible. The reason is that the meta-information of one variable of the hierarchical concept is stored in an HDF5 group, and HDF5 does not support compression of such an object.

The binary result file of the Modelica modeling and simulation environment Dymola [DS12] is used as a reference to compare the file sizes generated in HDF5. This proprietary storage format of Dymola is very compact. Dymola stores variables with different names and same data (so called alias variables) just once. Dymola also stores negated alias variables, i.e. the numeric data of two variables $a = -b$ only once. If b is stored, for variable a only the information is stored that it has the values of $-b$. This aliasing method is also used in the MTSF and in the hierarchical format.

As the performance measurements below indicate, storing many (more than 1000) objects in HDF5 with standard options is very storage consuming. The storage requirements can be considerably reduced by using the following two options [THG11]:

- For the storage strategy of objects the option *Compact* (in Python: `h5py.h5d.COMPACT`) should be used, instead of *Contiguous* or *Chunked*. This option leads to storing the raw data of small datasets in the header of the dataset.
- In HDF5 1.8.0 an optional mechanism is introduced to store groups much more efficiently by using a fractal heap and indexed with an improved B-tree. In order to activate this feature, the version number in which the HDF5 file is generated needs to be specified by the option *H5F_LIBVER_LATEST*. In Python, the file has to be opened by `h5py.File(..., libver="latest")`.

The full robot model (see section 3) is used as test case. This model has about 7000 variables, where 2500 are parameters and constants, 800 variables are time varying and the other variables are alias or negated alias variables. For the performance test 500 fixed grid result points and 2·50 varying grid result

points due to 50 state events are taken into account. Discrete variables are only stored at event points, but continuous variables are stored at grid and event points, here at 600 points. This gives the sizes of the files in Table 2.

Table 2: File sizes of the RobotR3 example for 500 grid points and different formats. The first column of *Relative Size* is normalized to the result of the Dymola format. The second column is normalized to the results of the MTSF format.

Format	Raw	MB	Relative Size	
			Normalized to Dymola	Normalized to MTSF
Hierarchical variables format with standard options	HDF5	27.6	5.11	7.46
Hierarchical variables format with options <i>compact</i> and <i>latest</i>	HDF5	7.4	1.37	2.00
Dymola format	MAT	5.4	1.00	1.46
MTSF format	HDF5	3.7	0.69	1.00

The MTSF format results in a file size that is just half of the file size of the HDF5 hierarchical variables format, so it is clearly superior. Furthermore, the MTSF format gives about 30% smaller file size with respect to the Dymola file, although more meta information is stored than in the Dymola file.

Table 3: File sizes for 5000 grid points and different formats. The first column of *Relative Size* is normalized to the result of the Dymola format. The second column is normalized to the results of the MTSF format.

Format	Raw	MB	Relative Size	
			Normalized to Dymola	Normalized to MTSF
Hierarchical variables format with standard options	HDF5	54.1	1.56	1.79
Hierarchical variables format with options <i>compact</i> and <i>latest</i>	HDF5	33.8	0.98	1.12
Dymola format	MAT	34.6	1.00	1.15
MTSF format	HDF5	30.2	0.87	1.00

For result files with increasing number of result points, the relative differences between the different

approaches is decreasing, which can be seen in Table 3 for 5000 fixed grid result points and 2·50 varying grid result points at events. The reason of a decreasing difference are the file size dominating data matrices that are identical at least in the HDF5 files.

4.3 Writing and Reading of Large Files

The previous tests evaluated writing of HDF5 files. If the HDF5 file becomes very large, it can no longer be read in one piece. Reading files which are larger than the main memory is slow, as virtual memory paging has to be used. The question arises how this is handled. In Dymola, and many other simulation programs, reading a result file requires to read it completely in to memory and then the file sizes that can be handled are restricted by the respective main memory. Here the power of the HDF5 format is applied. It is possible to read just a specified column of a matrix, without reading the whole matrix. Internally, the HDF5 matrix is split into chunks (= smaller matrices) and only the relevant chunks are read [THG11].

Performance of writing and reading some parts of a huge matrix depends on amongst others the sizes of the chunks. Because it is not fixed what parts of result matrices are read after writing, the chunking details are not specified for an MTSF file.

Table 4: File sizes and performances of writing and reading MTSF files.

# Rows	# Columns	MB	GB	Writing [s]	Reading 1 [s]	Reading 2 [s]
$6 \cdot 10^3$	766	35.5	0.03	0.5	0.15	0.02
$6 \cdot 10^4$	766	352	0.34	5.9	0.23	0.06
$6 \cdot 10^5$	766	3517	3.4	62.2	0.84	0.2
$6 \cdot 10^6$	766	35160	34	1109	4.9	1.2
$3.6 \cdot 10^7$	766	210410	205	11400	25.5	1.9

In Table 4 experiments with the full robot model on a solid state disk (on a system with an Intel Xeon X5550 @ 2.67 GHz processor) are documented. The number of time points has been increased to get large HDF5 files. Performance of reading two columns (time and one model variable) of the matrix *Re-*

sults/Continuous/H5T_NATIVE_DOUBLE into Python is documented in column *Reading 1*. Performance of reading the last row of the matrix (final value of all variables) is shown in column *Reading 2*. We did not investigate how different chunk sizes influence the result. It is clear that a fine tuning can improve the numbers in Table 4.

This test proves to be able to write data to and read it from result files beyond 200 GB in acceptable time. Further tests should verify the handling of huge files. Using HDFView, the structure of large files can be inspected without problems. Only for the 205 GB file, HDFView is slowing down.

4.4 Reading by Matlab

Matlab [M12] is one of the most commonly used scripting tools in engineering applications. Therefore it has to be simple and fast to read data from MTSF files in Matlab. The test concentrates on reading the names of all variables of a result file. Using this list of variables a variable tree browser could be generated. We investigate reading two files of the full robot model. One file is according to the proposed MTSF format (see Section 3), the other file follows the hierarchical variables concept (see Section 4.1).

Table 5: Time for reading all variable names in different formats. For the hierarchical variables concept we distinguish between a format that includes HDF5 enumerations in attributes of HDF5 datasets and replacing them by simple integer values.

Matlab function	Hierarchical Variables Concept		MTSF
	Enum.	Integer	
<code>h5info</code>	Error	75 s	0.1 s
<code>hdf5info (outdated)</code>	13 s	5.5 s	0.1 s

Matlab 2011b offers the high level functions `h5info` for reading the structure of an HDF5 file and `h5read` for reading one dataset. To get the names of all variables for the hierarchical variables concept one has to read the tree structure of the HDF5 group *Model-Variables*. We use `h5info` for it. Apparently, Matlab is not able to read enumeration attributes in HDF5 datasets. Therefore, we generated a new result file and replaced enumerations by simple integer values. The elapsed time for reading the different files are listed in Table 5. Using the outdated Matlab function

hdf5info we were able to reduce the elapsed time for the hierarchical variables concept.

The MTSF file contains only a few HDF5 groups and datasets, whereas the file of the hierarchical variables concept includes many (small) groups and objects. So it seems evident that reading the result file structure is faster for the MTSF file. To get all variable names from a MTSF file one has to read the dataset *ModelDescription/Variables*. Using the Matlab command `h5read('fullRobot.mtsf', '/Model-Description/Variables')` the information is available. The execution time for this command is 0.02 s. In summary, reading the variable names from the MTSF file is much faster than for the hierarchical variables concept. These preliminary tests with Matlab also clearly indicate that the proposed file format is better suited than the hierarchical variables concept. Furthermore, the Matlab `h5read` m-file does not support region references. Besides the other drawbacks discussed in Section 3.1, it is therefore advisable to not use region references in HDF5 files, if the files should be read by Matlab.

5 Conclusions

A standard for time series result files typically generated by dynamic model simulations is proposed. The standard is based on the HDF5 file format because HDF5 offers many features to flexibly and efficiently store data. In test cases huge files larger than 200 GB are successfully written and read. We hope to come into discussion with all persons who are interested in a standard result file format. The goal is to define an internationally well accepted standard that is supported by many tool vendors.

6 Acknowledgements

We acknowledge the coding and testing work of J. M. Solis Lopez' (formerly Bausch-Gall GmbH). M. Friedrich (Simpack AG) gave very useful information to reduce the file size of HDF5 files generated when using the hierarchical variables concept. We acknowledge his support. Also, we are grateful for the constructive comments of the reviewers.

References

- [A12] Association for Standardisation of Automation and Measuring Systems. www.asam.net.
- [BP11] Bausch-Gall I. and Pfeiffer A.: *Standard efficient Storage of Simulation Results*. ASIM2011, 21. Symposium Simulationstechnik, 7. - 9. Sept. 2011, Winterthur, Switzerland, 2011.
- [DS12] Dassault Systèmes AB: *Dymola*. www.dymola.com.
- [H12] H5py. <http://pypi.python.org/pypi/h5py>.
- [M12] MathWorks: *Matlab*. www.mathworks.com/products/matlab.
- [MA10] Modelica Association: *Modelica Standard Library 3.2*, Oct. 2010. www.modelica.org/libraries/Modelica.
- [MC10] MODELISAR consortium: *Functional Mock-up Interface for Model Exchange, Version 1.0*, 2010. www.functional-mockup-interface.org.
- [MC12] MODELISAR consortium: *Functional Mock-up Interface for Model Exchange and Co-Simulation, Version 2.0 Beta 3*, 2012. www.functional-mockup-interface.org.
- [NCS+12] http://access.ncsa.illinois.edu/Releases/05Releases/07.12.05_NCSA%27s_HDF.html
- [PHH+12] Pfeiffer A., Hellerer M., Hartweg S., Otter M. and Reiner M.: *PySimulator – A Simulation and Analysis Environment in Python with Plugin Infrastructure*. Accepted for publication in the Proceedings of 9th International Modelica Conference, Munich, Germany, Sept. 2012.
- [PA11] Phillips A. W. and Allemang R. J.: *Requirements for a Long-term Viable, Archive Data Format*. Structural Dynamics, Conference Proceedings of the Society for Experimental Mechanics Series, Volume 12, pp. 1475-1479, Springer, New York, 2011.
- [P10] Pointot, M.: *Five Good Reasons to Use the Hierarchical Data Format*. Computing in Science & Engineering, Vol. 12, Issue 5, pp. 84-90, 2010.
- [P12a] Python. www.python.org.
- [P12b] Python(x,y). www.pythonxy.com.
- [THG11] The HDF Group: *HDF5 User's Guide*, HDF5 Release 1.8.8, Nov. 2011.
- [THG12a] The HDF Group, www.hdfgroup.org.
- [THG12b] The HDF Group: *HDFView*. www.hdfgroup.org/hdf-java/html/hdfview.
- [TSC12] The Scilab Consortium: *Scilab*. www.scilab.org.

Towards a Memristor Model Library in Modelica

Kristin Majetta¹ Christoph Clauss¹ Torsten Schmidt²

¹Fraunhofer Institute for Integrated Circuits IIS, Design Automation Division EAS Dresden
Zeunerstrasse 38, D-01069 Dresden, GERMANY

²Technische Universität Dresden, Faculty of Electrical and Computer Engineering
Helmholtzstrasse 18, D-01062 Dresden, GERMANY

{kristin.majetta, christoph.clauss}@eas.iis.fraunhofer.de Torsten.Schmidt1@tu-dresden.de

Abstract

The Modelica realization of two memristor modeling approaches is presented which is compatible to the Modelica Electrical Analog Library. Circuit examples of some basic cases of application are simulated. Comparisons with published simulation results show the correctness of the numerical realizations. The models are the base of a Memristor Model Library.

Keywords: Memristor, window function, modeling, pinched hysteresis, resistive switches, numerical simulation of electronic devices

1 Introduction

The memristor is a special kind of resistor with memory. Therefore, the term “memristor” is composed by parts of both words “memory” and “resistor”. The theoretical concept of a memristor was published first in 1971 by Leon O. Chua [3]. After Strukow et al. [5] observed that certain nanoscale devices with thin semiconductor layers can be described as memristors, an intensive investigation started on both how a memristor works and how it can be utilized in electronic circuits. Since then, the memristor has a wide attention in the research community of electrical engineers, physicists, and biologists. Recent investigations are mainly focused on resistive random access memories. The advantage of memristors (or more general of memristive devices) is to store information without any power source to be needed. This could open a new paradigm in power saving computation as well as low power storage. Other fields of research are neuromorphic systems, memristor circuits theory, and applied analog memristor circuits. It can be expected that further interesting fields of both research and application will be opened up in future.

Simulation has been applied since the very beginning of integrated circuit development, so it does for memristor circuits. Therefore, memristor modeling became necessary, and simulation models were published which can be used in different simulation tools. E.g. MATLAB models use a state equation based approach, but models for SPICE need a combination of built-in SPICE models which realize the memristor behavior.

This paper deals with the adaption of published memristor models to Modelica. It is the first step towards the general aim to create a library for memristors, and memristive systems (memcapacitors, meminductors, memristive systems with more than one state [6]). The library will allow to investigate memristor applications on the one hand on circuit level, and on the other hand in the context of arbitrary Modelica applications. In section 2 two different models are presented. Simulation results using Dymola are shown in section 3.

2 Model Approaches

Once memristor measurement data were available several model approaches were elaborated, and adapted to the data. Two very basic models are presented in this section.

2.1 A Basic Model Approach on Varying Resistance

Basing on the physical device structure the authors of [1], [2], [5] introduce a memristor model as a resistor with varying resistance R_{MEM}

$$v(t) = R_{MEM}(x)i(t) \quad (1)$$

which depends on x linearly and changes between R_{ON} and R_{OFF} :

$$R_{MEM}(x) = R_{ON}x + R_{OFF}(1-x) \quad (2)$$

The structure of such a device is depicted in figure 1 and consists of a partly doped TiO_2 -layer which is together with the undoped part sandwiched between two Pt-electrodes. With w being the length of the doped region (Figure 1), and D the total length of doped and undoped region, the state x is defined as

$$x = w/D \quad (3)$$

The doped region is highly conducting (R_{ON}) whereas the undoped region is less conducting (R_{OFF}). Taking into account the length of both regions equation (2) represents a series connection of the actual resistances of both the doped and undoped region.

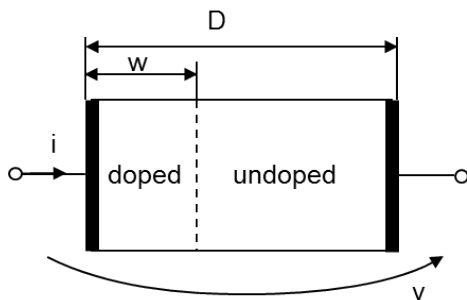


Figure 1 Physical Memristor Schematic

The state equation (4) describes the speed of the boundary movement, which depends on the current $i(t)$, the resistance R_{ON} of the doped region, the dopant mobility μ_v and a window function $f(x)$:

$$\frac{dx}{dt} = \frac{\mu_v R_{ON}}{D^2} i(t) f(x) \quad (4)$$

The so-called window function establishes a nonlinear drift behavior of the device and hence provides a second order approximation to the real device. According [3] one possible window function is

$$f_{Jog}(x) = 1 - (2x - 1)^{2P} \quad (5)$$

whereas the authors of [1] propose an improved window function f_{Bio} which overcomes sticking of x at 0 or 1 whenever x reaches one of these values. The proposed window function depends on both x and the sign of the current $i(t)$:

$$f_{Bio}(x) = 1 - (x - stp(-i(t)))^{2P} \quad (6)$$

$$stp(i(t)) = \begin{cases} 1 & i(t) \geq 0 \\ 0 & i(t) < 0 \end{cases} \quad (7)$$

The equations (1) to (7) are the base for writing a simulation model. One possibility is constructing a SPICE subcircuit out of SPICE basic components (macromodeling) according to [1]. This subcircuit could be copied to Modelica using the Modelica.Electrical.Spice3 package. A more convenient way is using the declarative behavioral modeling capability of Modelica. This leads straight forward to the following model, called Memristor_Biolek2009:

```

model Memristor_Biolek2009
import ME = Modelica.Electrical;
import SI = Modelica.SIunits;
extends
    ME.Analog.Interfaces.OnePort;
parameter SI.Resistance RINIT, RON,
    ROFF;
parameter SI.Length D;
parameter Real muev;
parameter Integer P;
SI.Resistance
    RMEM(start=RINIT, fixed=true);
SI.Length w;
Real x, k, f;
equation
RMEM = RON*x + ROFF*(1-x);
x = w/D;
v = RMEM*i;
der(x) = k*i*fBio;
k = (muev*RON) / (D^2);
//fJog = 1 - (2*x-1)^(2*P);
fBio = 1 - (x - stp(-i))^(2*P);
end Memristor_Biolek2009

```

The special sign function is

```

function stp
input Modelica.SIunits.Current i;
output Real value;
algorithm
    value:=if (i<0) then 0 else 1;
end stp;

```

An icon, default values for parameters as well as assertions complete the model.

2.2 An Improved Approach

The authors of [6] propose an improved approach with more parameters than the model already presented which can be better adapted to given memristor characteristics, e.g. measured data.

Like the Memristor_Biolek2009 model this model has a state variable $x(t)$ for calculating the conductivity according to

$$i(t) = \begin{cases} a_1 x \sinh(bv(t)) & v(t) \geq 0 \\ a_2 x \sinh(bv(t)) & v(t) < 0 \end{cases} \quad (8)$$

with a_1 , a_2 , and b being adjustable parameters. The state equation for x is:

$$\frac{dx}{dt} = g(v(t))f(x) \quad (9)$$

Whereas $g(t)$ is a threshold function which ensures a state changes only if thresholds are exceeded:

$$g(v(t)) = \begin{cases} A_p (e^{v(t)} - e^{v_p}) & v(t) > v_p \\ -A_n (e^{-v(t)} - e^{-v_n}) & v(t) < -v_n \\ 0 & \text{else} \end{cases} \quad (10)$$

The window function $f(x)$ expresses the effect that it is harder to change the state near the boundaries, taking into account the polarity. Parameters are introduced to be able to fit to measured values.

$$f(x) = \begin{cases} e^{\alpha_p(x-x_p)} w_p(x, x_p) & x > x_p \\ e^{\alpha_n(x+x_n-1)} w_n(x, x_n) & x < 1-x_n \\ 0 & \text{else} \end{cases} \quad (11)$$

$$w_p(x, x_p) = \frac{x_p - x}{1 - x_p} + 1 \quad (12)$$

$$w_n(x, x_n) = \frac{x}{1 - x_n} \quad (13)$$

The equations (8) to (13) can be formulated in Modelica as they are. This leads to the second memristor model, called Memristor_Yakopcic2011:

```

model Memristor_Yakopcic2011
import ME = Modelica.Electrical;
import SI = Modelica.SIunits;
extends
ME.Analog.Interfaces.OnePort;
parameter Real Ap, An;
parameter SI.Voltage Vp, Vn;
parameter Real xp, xn, ap, an;
parameter SI.Current a1, a2;
parameter Real xinit;
parameter SI.InversePotential b;
Real gV, fx, wp, wn;
Real x (start=xinit, fixed=true);
equation
i = if (v>=0) then a1*x*sinh(b*v)
      else a2*x*sinh(b*v);
gV = if (v>Vp) then Ap*(exp(v) -
      exp(Vp))
      elseif (v<-Vn) then
      -An*(exp(-v) - exp(Vn))
      else 0;
fx = if (v>0 and x>= xp) then
      exp(-ap*(x - xp))*wp
      elseif (v>0 and x<xp) then 1
      elseif (v<0 and x<=1-xn) then
      exp(an*(x+xn-1))*wn
      else 1;
wp = (xp - x)/(1 - xp) + 1;
wn = x/(1 - xn);
der(x) = gV*fx;
end Memristor_Yakopcic2011
    
```

3 Test And Application Examples

3.1 Memristor Characteristic Using One Input Voltage Pulse

The first example shows the Memristor_Biolek2009 characteristic using a simple voltage pulse.

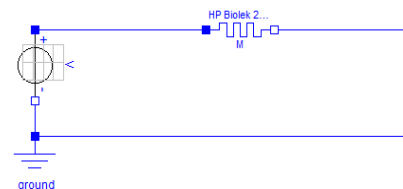


Figure 2 Memristor test circuit

RINIT	RON	ROFF	D	muev	p
11000	100	16000	1e-8	1e-14	10

If the above mentioned parameters are used, the voltage pulse of Figure 3 causes the current-voltage hysteresis of Figure 4. The reason for the hysteresis is increasing of the doped region length as long as a positive current is flowing which increases the overall resistance. Figure 5 shows the change of the state x which influences the resistance. This illustrates that the state is the “memory” of the memristor. The initial state is caused by the initial value RINIT. No special hysteresis model is used, only changing the state causes the hysteresis.

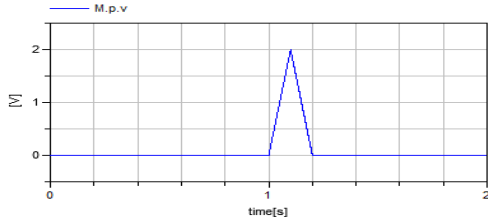


Figure 3 Single voltage pulse

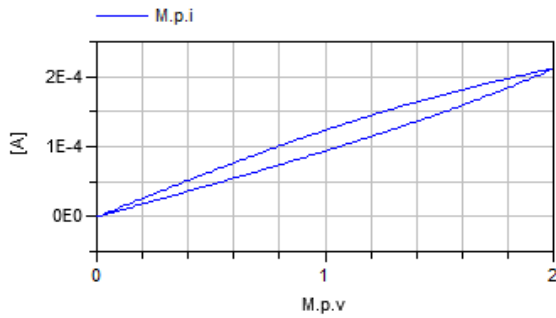


Figure 4 Current-Voltage Hysteresis

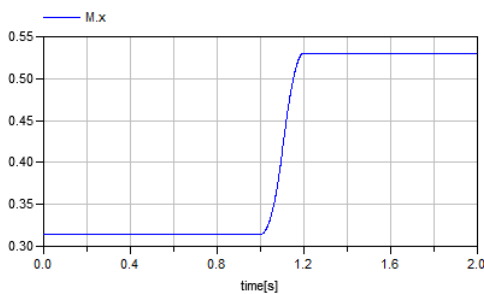


Figure 5 State change due to the voltage pulse

3.2 Memristor Characteristic Using Sinusoidal Input Voltage

To compare the Memristor_Biolek2009 characteristic with the results published in [1] a circuit like in Figure 2 is simulated using a sinusoidal voltage input (1.2 V amplitude, 1 Hz). The memristor parameters are the same as in the example 3.1. Figure 6 shows both the input voltage, and the resulting current.

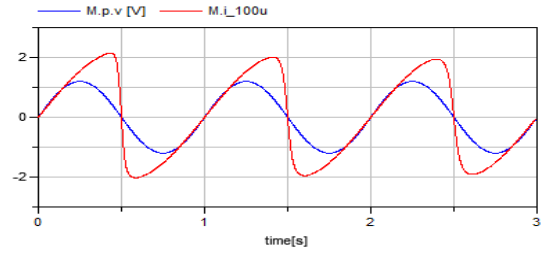


Figure 6 Input voltage and current (raised by 100), with RINIT=11000 Ohm

This result differs slightly from the result published in [1] depicted in Figure 7 due to initial transient effects. In the steady state which can be reached by changing the initial resistance to RINIT=11500 or by simulating over a long time period the published results are reached (Figure 8, Figure 9). The reason for these differences in initialization can depend on different numerical algorithms, and on different error bounds. This has to be investigated in future.

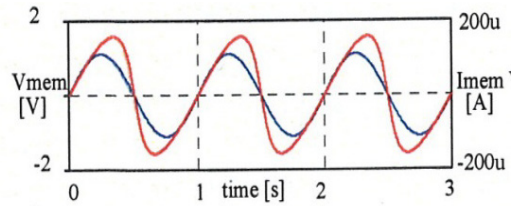


Figure 7 Reference result according to Figure 6, published in [1] (input voltage and current)

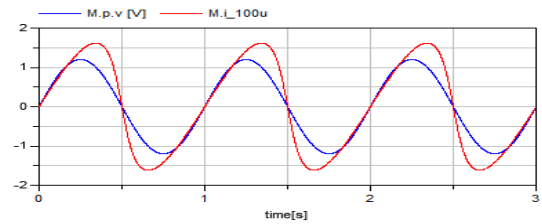


Figure 8 Input voltage and current (raised by 100), with RINIT=11500 Ohm

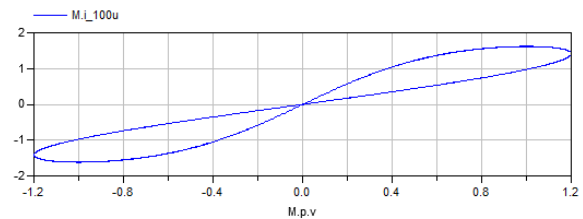


Figure 9 Current-voltage characteristics (current raised by 100), with RINIT=11500 Ohm

Due to the Memristor_Biolek2009 model, formula (7), the window function $f(x)$ is discontinuous. According to (4) jumping of $f(x)$ influences the derivative of x but not x itself. Therefore, such discontinuities did not yet lead to simulation difficulties in the investigated examples.

3.3 Memristor Characteristic Using Multiple Input Voltage Pulses

The simple circuit according to Figure 2 is used to check the characteristic of the Memristor_Yakopcic2011 model. The results are compared with Fig. 4 in [5]. The memristor model parameters are:

ap	an	Ap	An	Vp	Vn
1	4	0.1	10	0.9	0.2
xp	xn	a1	a2	b	xinit
0.15	0.25	0.076	0.06	3	0.001

The following simulation results (Figure 10, Figure 11) achieved with Dymola are the same as in the reference simulation. The memory effect can be seen. According to this first check the model seems to be correct.

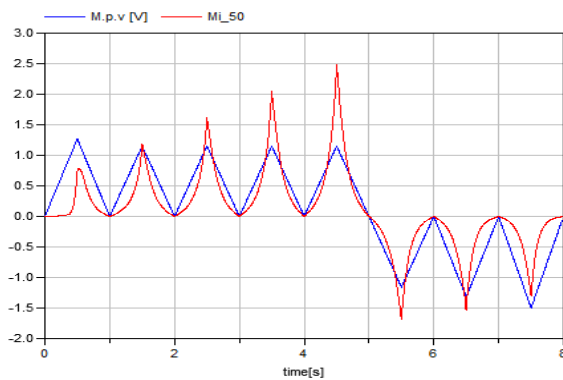


Figure 10 Current-voltage characteristic (current raised by 50), Jakopcic2011 model

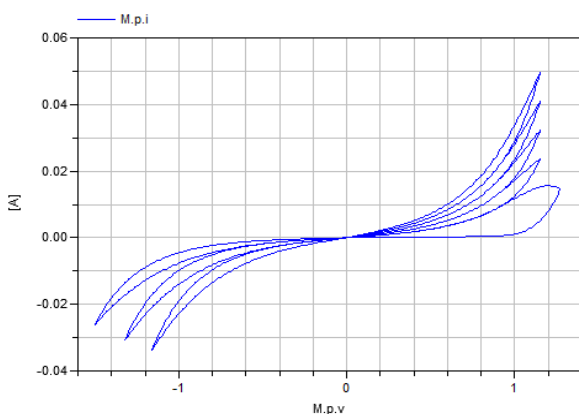


Figure 11 Current-voltage hysteresis, Jakopcic model

3.4 Graetz Rectifier Circuit

Figure 12 shows a Graetz rectifier circuit which uses memristors instead of diodes. It is easily combined using the presented memristor_Yakopcic2011 model as well as MSL components. The memristor parameters are the same as in the previous section.

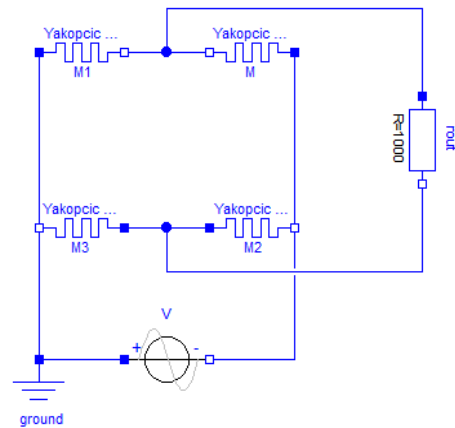


Figure 12 Graetz circuit using memristors

Both the rectified and the original voltage can be seen in Figure 13. Deeper investigation shows that the amplitude of the rectified voltage depends on the amplitude of the input voltage as well as on the frequency. Higher frequency causes smaller rectified voltage amplitudes. If the input voltage is too high the simulation fails. The reasons of that seems to be extremely increasing of exponential functions. Therefore, the model must be improved in future to become more stable.

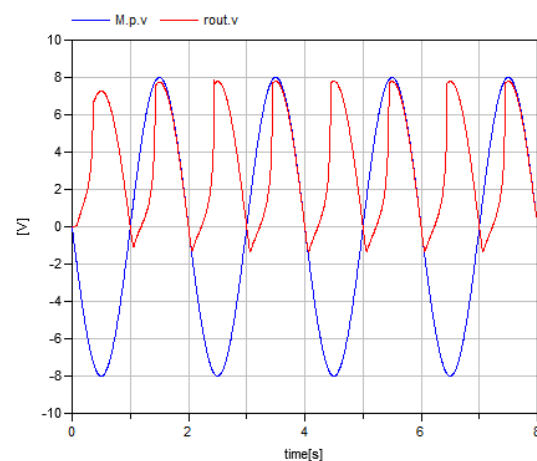


Figure 13 Input voltage, and rectified voltage

4 Conclusions

Two memristor models developed from given memristor model equations are presented. Simple tests show the correctness of the models compared with published simulation results. In some cases small differences occur that have to be investigated in future. Tests with different simulation tools are still necessary, which cover extreme application scenarios.

The Modelica approach of memristor modeling is promising. The memristor models can easily be combined to existing models of the Modelica Standard Library. It is planned to develop a package with numerical stable, and well tested models of memristors, and of other memristive systems like memcapacitors, meminductors, or systems with more than one states. This will allow to study memristors and memristor application circuits in a convenient way.

References

- [1] Biolek, Z.; Biolek, D.; Biolkova, V.: Spice Model of Memristor With Nonlinear Dopant Drift. *Radioengineering*, vol. 18, no. 2, pp. 210-214, 2009.
- [2] Biolkova, V.; Kolka, V.; Biolek, Z.; Biolek, D.: Memristor modeling based on its constitutive relation. *Proc. of the European Conference of Circuits Technology and Devices (ECCTD)*, pp. 261-261, 2010.
- [3] Chua, L. O.: Memristor – the missing circuit element. *IEEE Trans. Circuit Theory* vol. 18, no. 5, pp. 507-519, 1971.
- [4] Joglekar, Y.; Wolf, S.: The elusive memristor: properties of basic electrical circuits. *Eur. J. Phy.*, vol. 30, pp. 661-675, 2009.
- [5] Strukow, D.; Snider, D.; Stewart, D.; Williams, R.: The missing memristor found. *Nature*, vol. 453, pp. 80-83, Mai 2008.
- [6] Ventra, M. D.; Pershin, Y. V.; Chua, L. O.: Circuit elements with memory: memristors, memcapacitors and meminductors. *Proceedings of the IEEE 97*, 1717 (2009), über arXiv:0901.3682v1 [cond-mat.mes-hall]
- [7] Yakopcic, C.; Taha, T. M.; Subramanyam, G.; Pino, R. E.; Rogers, S.: A Memristor Device Model. *IEEE ELECTRON DEVICE LETTERS*, vol. 32, pp 1436-38, Oct. 2011.

Fault Detection of Power Electronic Circuit using Wavelet Analysis in Modelica

Jianbo Gao*, Yang Ji**, Johann Bals**, Ralph Kennel*

*Technische Universität München

Arcisstr. 21, 80333 Munich, Germany

michael.gao@tum.de, ralph.kennel@tum.de

** German Aerospace Center

Muenchner Str. 20, 82234 Wessling, Germany

yang.ji@dlr.de, johann.bals@dlr.de

Abstract

In more electric aircrafts (MEA) the electric power network is important for the reliability. To prevent severe faults it is the key issue to identify the faults in the early stage before a complete failure happens. In this paper an early stage fault detection method using wavelet multi-resolution analysis (MRA) for a regulated buck DC-DC converter is studied. Specifically, the electrolyte input capacitor is diagnosed. The study was carried out using simulation with Modelica / Dymola. The fault features that were extracted from different levels of wavelet decomposition provided clear information for both fast and slow occurring faults. This method showed significant advantages compared with filter techniques. It is concluded that wavelet transform is a suitable tool for early stage fault detection of the power electronics in MEA. In addition, the simulation language Modelica provides a convenient possibility for the quick design of fault detection strategy.

Keywords: power electronics; DC-DC converter; fault detection; wavelet; Modelica; Dymola

1 Introduction

1.1 Motivation

The concept of More Electric Aircraft (MEA) is attracting increasing interest in the aircraft industry not only because of its potential in energy optimization, but also due to its significant advantages concerning weight, maintenance requirements, liability and passenger comfort [1]. For this, the electrical power distribution network is playing a more important role and facing increasing challenges in the prognosis and accurate localization of faulty units in an even more complex power network. In order to

obtain maximum flight reliability and minimum maintenance efforts, advanced failure analysis technologies shall be applied to ensure correct and quick fault detection and isolation. It is well known that an output voltage regulated DC/DC power converter supplying constant power loads could de-stabilize the network stability due to the degraded performance of its input filter. The sensitivity study of input filter parameters concerning the network stability addressed in [2] reveals that the observation of degraded degree of the capacitor in the input filter can significantly increase the network reliability.

1.2 State of the art

Reviewing the considerable development of fault diagnosis techniques and many successful applications attached to them in the last time [3] [4] [5], power systems keep a challenge for fault detection. For overcoming this challenge intelligent methods like artificial neural networks have shown their possibilities in this field [6]. Besides, the analytical model based technology is also obtaining more attention [7] [8] [9].

Signal-based methods, e.g. Fourier transform and wavelet transform, also provide other possibilities to perform the fault detection and isolation. With the rash development of the new mathematical tool, wavelet transform [10], a great amount of studies have been done in different fields for fault detection. Some attempts have also been made in power electronics for fault detection [11] [12]. The implementation of wavelet transform for the post processing of Modelica simulation data has been seen, for example, in a study of vehicle steering, where wavelet transform was carried out in the software Matlab for calculating power spectra [13].

1.3 Main contributions

Modelica was developed as a free, object-oriented and equation-based modeling language. It has significant benefits such as easy reusability of models and multi-domain modeling capability. In combination with the simulation environment Dymola, a convenient platform is provided to the complete model-based design and the integration of MEA systems [14]. In contrast to the excellent performance in modeling and simulation, Modelica only supports limited signal analysis features [15], which are actually crucial for the fault analysis and virtual testing activities in the verification and validation phase of the system development.

This work focuses on the fast design of a fault detection strategy of on board power supply units in MEA with wavelet transform using Modelica simulation. To realize this, a wavelet library for Modelica is being developed. Multi-Resolution Analysis (MRA) of wavelet technology is applied to detect the failure of electrolyte capacitors in a very early stage.

In addition, the design process using Modelica simulation shows high flexibility and efficiency. It is possible to identify the most important failure features and helps to design a effective fault detection strategy within only a short time.

2 Wavelet transform

2.1 Definition

Wavelet transform could be considered as a further development of Fourier transform, or more precisely, of short time Fourier transform (STFT) [16]. Using STFT, people try to localize the signal changing with time by selecting suitable time window. This transformation, however, is limited in time-frequency resolution capability due to the uncertainty principle. Wavelet transform overcomes this problem. This transform is defined as [10]:

$$[W_{\psi}f](a, b) = \frac{1}{\sqrt{|a|}} \int_{-\infty}^{\infty} \overline{\psi\left(\frac{t-b}{a}\right)} f(t) dt. \quad (1)$$

It is described as the wavelet transform of the square-integrable function, f , using wavelet function, ψ , at dilation (or scale), a , and position (or translation), b . The bar above function, ψ , stands for conjugation. For the given a and b , the transform result is a single real number, a wavelet coefficient.

Obviously wavelet transform is the integral of the multiplication of the signal, f , with a wavelet function, ψ . It has the same form as the STFT. However, not like STFT, where only sine and cosine functions

are used for the transformation, wavelet transform uses different wavelet functions, which can be selected according to the specific problems from a principally unlimited set. Nevertheless, the wavelet function must fulfill some conditions; namely, it must be an orthonormal function. The precise mathematical description of orthonormality is easily found in almost every book about wavelet transform, e.g. [10], and is not repeated here.

Parameter, a , defines the width and height of the wavelet function, ψ . A smaller scale, a , makes ψ narrower; thus the wavelet represents fast changes and the transform focuses on the high frequency components of the signal. Parameter, b , shifts the wavelet function along the time axis, so that the transform represents different locations of the signal. Using different values of scale, a , and position, b , it is able to observe the signal at different position and in different frequency range with only one transformation. Thanks to these special properties, wavelet transform is especially suitable for analyzing changing processes.

Two forms of wavelet transform are available. They are continuous wavelet transform (CWT) and discrete wavelet transform (DWT). In CWT both scale and position parameters are continuous real values. CWT expresses the signal changes in continuous details. It is more suitable for visual examination. In this work only DWT is used, which will be described in more detail in the next section.

2.2 Discrete wavelet transform

In DWT only discrete values of the scale and location parameters are used. The values are selected in a discrete form, namely

$$\psi_{j,k} = 2^{-\frac{j}{2}} \psi(a^{-j}x - kb), \quad (2)$$

where $a > 1$, $b > 0$ and $j, k \in \mathbf{Z}$. The transform results, i.e. the wavelet coefficients, are therefore also discrete.

In the numeric calculation of DWT, an extra scaling function, in addition to the wavelet function, is used to carry out a complete and efficient DWT. The scaling function represents the low frequency components of the signal. It is orthogonal to its own discrete translations and to all wavelet functions. The wavelet and the scaling functions with the discrete scaling and translation parameters build a complete orthogonal basis of the Hilbert space. The DWT is thus another representation of the time signal.

As an example, Figure 1 shows the form of the third order Daubechies scaling and wavelet functions and their Fourier transforms [17].

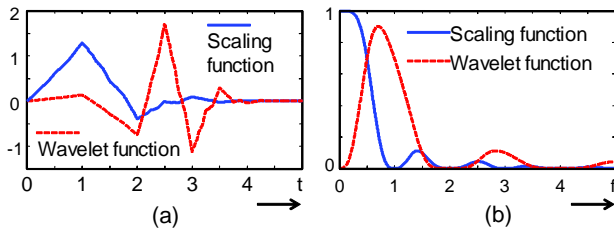


Figure 1: The third order Daubechies scaling and wavelet functions (a) and their Fourier transforms (b)

From the Fourier transforms it can be seen that the scaling function covers lower frequency range while the wavelet function stretches in a higher frequency range. From this point of view, DWT is actually the division of the time signal into different frequency bands. Thus, it is straightforward to understand that the calculation of DWT is realized using filter banks. In inverse DWT the calculation is similar. This process is illustrated in Figure 2.

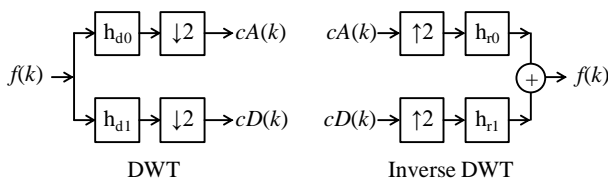


Figure 2: DWT and inverse DWT calculation using filter banks

DWT transforms the original sequence in two new series:

- (1) the approximation coefficients, $cA(k)$, representing the low frequency components, obtained using the low pass filter for decomposition, h_{d0} , and
- (2) the detail coefficients, $cD(k)$, representing the high frequency components, obtained using the high pass filter for decomposition, h_{d1} .

The symbol $\downarrow 2$ means down sampling. The operation is to delete one from every two adjacent coefficients, in order to remove the redundant information. The inverse DWT carries out the reversed operation. The operator, $\uparrow 2$, expands a coefficient series by inserting a zero between every two adjacent elements. After that the two series pass through the filter bank, and added together to get the original signal.

2.3 Multi-resolution analysis

Considering the DWT process shown in Figure 2, sequence, $cA(k)$, which represents the low frequency components can be further divided into a lower frequency part and a higher frequency part inside the frequency range of $cA(k)$. This process is repeated and a series of coefficient sequences representing

different frequency ranges is obtained, as shown in Figure 3:

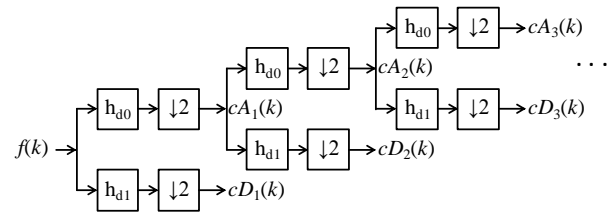


Figure 3: Multi-resolution analysis using DWT

This is the wavelet multi-resolution analysis (MRA). The output of this operation, cD_1, cD_2, \dots, cD_n and cA_n , are different levels of DWT coefficients, representing the signal components from higher to lower frequencies. Here the original signal is treated as the lowest level of approximation coefficients. This analysis provides a convenient tool to observe different frequency components of the signal depending on time.

2.4 Wavelet analysis for fault detection

Wavelet transform is a powerful tool in signal processing for the detection of changing events. This feature is suitable for fault detection since a fault in a system can be treated as a deviation compared to the normal state.

When a fault occurs, specific changes will appear in the sensor signal. Usually, it is known that the fault signal is located in a certain frequency range, but the exact frequency is often unknown or not constant. This problem can be handled with wavelet MRA. For that, the signal containing fault information is firstly decomposed in several levels. And in one or more levels, where the fault signal frequency is located, faults features will be observed.

3 Wavelet fault detection in a MEA power network system

Based on the properties of wavelet transform in signal processing, this new mathematic tool is used for fault detection in a MEA power network system in this study. Specifically, the MRA is used here to detect the capacitance drop of the input capacitor in a DC-DC buck converter for the early stage failure.

3.1 The problem

The buck converter is described with the diagram shown in Figure 4.

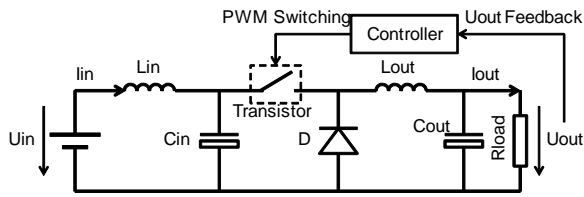


Figure 4: Diagram of the Buck converter under study

The converter operates in constant load power mode by keeping a constant output voltage, which is adjusted through the duty cycle of the pulse width modulation (PWM) switching signal, which operates with a constant frequency. Based on the converter property the system is sensible to the value changes of the components on the input side, where the input capacitor, C_{in} , is especially critical because it is usually an electrolyte type, which has significantly lower feasibility and shorter life time compared with other components. Based on this reason, the early fault detection is focused on C_{in} .

Four parameters of the circuit can be conveniently measured by voltage and current sensors. They are, referring to Figure 4, U_{out} , I_{out} , I_{in} and U_{in} , ordered from higher to lower ease of availability. Since the load is pure resistive constant load, the output current signal will contain the same information as the output voltage. And U_{out} will be used any way for the controller as feedback, the equipment of a sensor for output current is therefore not necessary, at the least for fault detection.

Since a stable output voltage is the control objective of the circuit, the influence of C_{in} would be compensated by the controller very quickly. As a consequence only very few fault information would be propagated to the output side. The fault detection using U_{out} is therefore not feasible.

The input voltage is not a good signal for fault detection, too, since it actually measures the input power supply voltage, which is normally a voltage source with very low impedance and thus hardly be influenced by C_{in} .

The input current is the last possibility; and it is actually also a suitable signal for fault detection. The reason is, for example, if its capacitance drops, it means the energy capacity of the input circuit is reduced. In order to keep a constant energy flow to the load, which is regulated by the controller, C_{in} would have to be charged and discharged more deeply. This will be reflected in the input current with larger fluctuation. This estimation will be showed later in the result section.

If an electrolyte capacitor approaches its life end, its capacity would reduce slowly within a certain time. However, sharp reduction or changing of ca-

pacitance might also occur. For fault detection, especially for early stage fault diagnosis, both stepping type and slow changing of capacitor fault should be considered.

3.2 Extraction of fault information

The first step is the extraction of the fault information from the sensor signals. Supposing the measured signal is

$$f(k) = x(k) + g(k), \quad (3)$$

where $x(k)$ is the signal in normal operation state, and $g(k)$ the additional signal in fault condition.

Using wavelet technology, the sensor signal $f(k)$ is decomposed with MRA using wavelet function, ψ , to obtain wavelet coefficients in n levels:

$$\begin{aligned} DWT_{\psi}\{f\} &= A_n\{f\} + \sum_{i=1}^n D_i\{f\}, \\ &= A_n\{x\} + A_n\{g\} \\ &\quad + \sum_{i=1}^n (D_i\{x\} + D_i\{g\}), \end{aligned} \quad (4)$$

where $D_i\{\cdot\}$ represents the detail coefficients, and $A_i\{\cdot\}$ stands for approximation coefficients. The term $D_i\{\cdot\}$ with smaller index, i , represents higher frequency components, namely faster changing signals.

The signal, x , in the normal operation condition is composed of the average value of the battery current, which changes very slow, and the ripples caused by the PWM controlling, which have a constant frequency defined by the controller. The slower components are transformed to the approximation coefficients, $A_n\{x\}$; and the components with PWM frequency, which are higher frequency components, is transformed to the very low level of $D_i\{x\}$.

On the other hand, the fault signal, i.e. the information of the reduction of C_{in} , is reflected by the fluctuation of the input current. As it is known that the fluctuation frequency is actually the PWM frequency, most of the fault information is contained in the PWM components, which means the lower levels of detail coefficients, $D_i\{g\}$. Depending on the fault occurrence rate, complex fluctuation could take place. This information will be carried by the PWM frequency, too, but its own frequency components are visible in other levels of $D_i\{g\}$.

In any circumstance few information will be present in very low frequency range, i.e. in $A_n\{g\}$. Therefore, we can extract the fault information from the input current signal simply by isolating some levels of detail coefficients, $D_i\{f\}$. Of course, the PWM information will also be involved. It has to be removed before the faults can be identified. Since this frequency is known and it always has a very high value, these components can be easily suppressed with low pass filter or band stop filter.

3.3 Fault identification

After all irrelevant information is excluded the final fault information is represented with a single value, changing according to the failure rate. The fault can be simply identified by comparing it with a known threshold.

4 Design of a fault detection strategy using Modelica

Because of the aforementioned superior properties, Modelica simulation and wavelet transform were selected for the quick design of a fault detection strategy for the power system in MEA. Since wavelet transform is not available in the standard Modelica libraries, a solution have to be found. A seemingly direct solution would be the use of a second software tool, which provides wavelet analysis capacity, such as Matlab from MathWorks. Some practical reasons were faced, however. Firstly, the use of such commercial software requires expensive licenses. Secondly, a single program both for simulation and data analysis is very desirable during the work in order to have an integrated working process and to avoid interfacing between two programs. It is therefore more favourable to have the wavelet analysis inside Modelica. In addition, this brings further advantages in that the library can be a common tool of Modelica, so that higher work efficiency will be achieved in a long term.

4.1 Model of the power supply

The buck converter shown in Figure 4 is realized with a Dymola model in Figure 5.

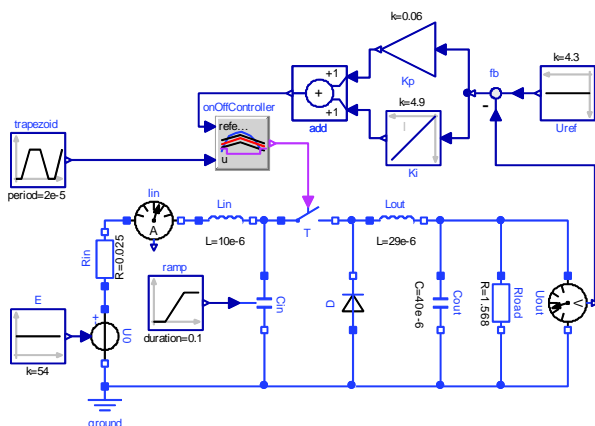


Figure 5: The Dymola model of the Buck converter for MEA

The voltage controller is a proportional-integral regulator. The output voltage is set as 4.3 V. The PWM frequency is defined with the trapezoidal

source as 50 kHz. The component fault is simulated by reducing the value of the input capacitor, C_{in} , with a ramp source. By setting the ramp, different changing rate of the component value can be simulated. The input current is measured by a current sensor, which is explicitly put in the model only for clarity, since the current values could actually be read out directly from the corresponding components, e.g. R_{in} or L_{in} . Other parameters are listed in Table 1

Table 1: Parameters of the buck converter

Parameter	Description	Value
E	Voltage source	54 V
U_{ref}	Reference output voltage	4.3 V
L_{out}	Output inductance	29 μ H
C_{out}	Output capacitance	40 μ F
R_{load}	Load resistance	1.568 Ω
L_{in}	Input filter inductance	10 μ H
C_{in}	Input filter capacitance	10 μ F
R_{in}	Input resistance	0.025 Ω
K_p	Propotional controller gain	0.06
K_i	Integral controller gain	4.9

4.2 Wavelet transform in Modelica

The structure of the wavelet toolbox developed for Modelica is shown in Figure 6. At the moment of this report the wavelet library is under development within the frame of Clean-Sky project organized by European Union. It is expected to be a general Modelica library with wavelet transform and some related functionality for different signal processing purposes. This library is used for post processing of the simulation result data and cannot be embedded into simulation models. So far the core library functions have been realized and wavelet DWT and MRA can be implemented for the reported work.

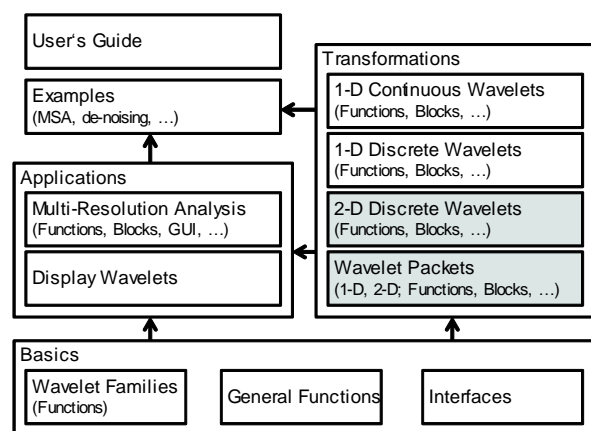


Figure 6: Structure of the intended Modelica wavelet library (the functions with dark background will be designed depending on the work process)

4.3 Process for simulation and fault detection

As mentioned in the problem description (section 3.1) the input current signal is used for fault detection. At first the model will be simulated with different reduction rates of the C_{in} capacitance. The reduction rates were selected between 1 and 100 ms with a capacitance drop from 10 to 8 μF , corresponding to 20% capacitance loss. With this fault level the system can still operate normally. However, a 20% reduction indicates a high possibility of a complete failure of the capacitor in the near future. The reduction is applied at 0.1 s after the startup of the simulation so that the system can achieve a stable state before the faults could occur.

After simulation, the data segment containing the fault event will be read out from the simulation result data file. It is converted to equidistant time series with a sampling rate of 200 kHz. Equidistant sampling is the requirement of wavelet transform and most other signal processing methods. The fault detection process is illustrated in Figure 7.

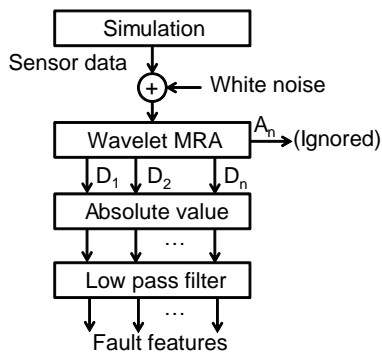


Figure 7: Process for simulation and fault detection using wavelet transform

To simulate the real world, a white noise signal with normal distribution is added on the input current signal. After that, MRA is applied on the data. The wavelet function used here was the third order Daubechies function shown in Figure 1. The detail coefficients in the DWT result are extracted and their absolute values are calculated since only the magnitude of the DWT coefficients contains fault information. To remove the high frequency PWM component, second order Butterworth low pass filter is used. The filter cut off frequency is set as 0.5 kHz, much lower than that of the PWM frequency, in order to suppress a large part of the noise signal, too. After this step, different fault features can be visually identified and suitable fault detection methods can be established.

4.4 Results

The tests with different parameters were carried out. Figure 8 and Figure 9 give two examples with slow and fast changing faults, respectively. Since all fault features mainly present in the first three levels of the wavelet decomposition, only these coefficients are shown in the figures.

It was noticed that the fault features differed significantly between fast and slow changing rates. For the changing rates faster than 5 ms, a pulse feature appeared in almost every level of the wavelet detail coefficients. This is well seen in Figure 8, where 20% capacitance drop took place within 1 ms. In the first MRA level the feature magnitude changed from 55 to 70 mA with the fault. In the other two higher levels the fault event was extra present with pulses. The magnitude of the pulses increased with the changing rate. The pulses appeared in other higher decomposition level, too. However, in level three it was significantly higher than those in other levels. For the fault with 1 ms dropping time, as shown in Figure 8, the pulse peak reached almost 100 mA in level 3. The simulation showed that for the dropping time up to 5 ms, the peak height reduced linearly to about 20 mA.

For the changing rates slower than 5 ms, the fault features were only significant in the lowest level of the wavelet MRA coefficients. Figure 9 shows an example, where the 20% capacitance drop took 100 ms time. Now only the first wavelet decomposition level contained a fault feature: the feature magnitude changed from about 55 to 70 mA linearly with the reduction of the capacitance. This was the same as the level 1 feature in fast changing faults. For even lower changing rates this linear relationship was always present. Not like in the case of fast changing fault, no significant features were observed in other decomposition levels.

Based on the simulation study, the following fault detection strategy can be defined for early stage fault detection of the input capacitor, C_{in} :

- (1) In wavelet decomposition level one, if the feature magnitude exceeds 70 mA, a capacitance reduction of 20% of C_{in} is detected.
- (2) In wavelet decomposition level three, if the pulse value exceeds 20 mA, the event implies a fast drop of the C_{in} capacitance. The pulse peak value could be used to estimate the capacitance changing rate.

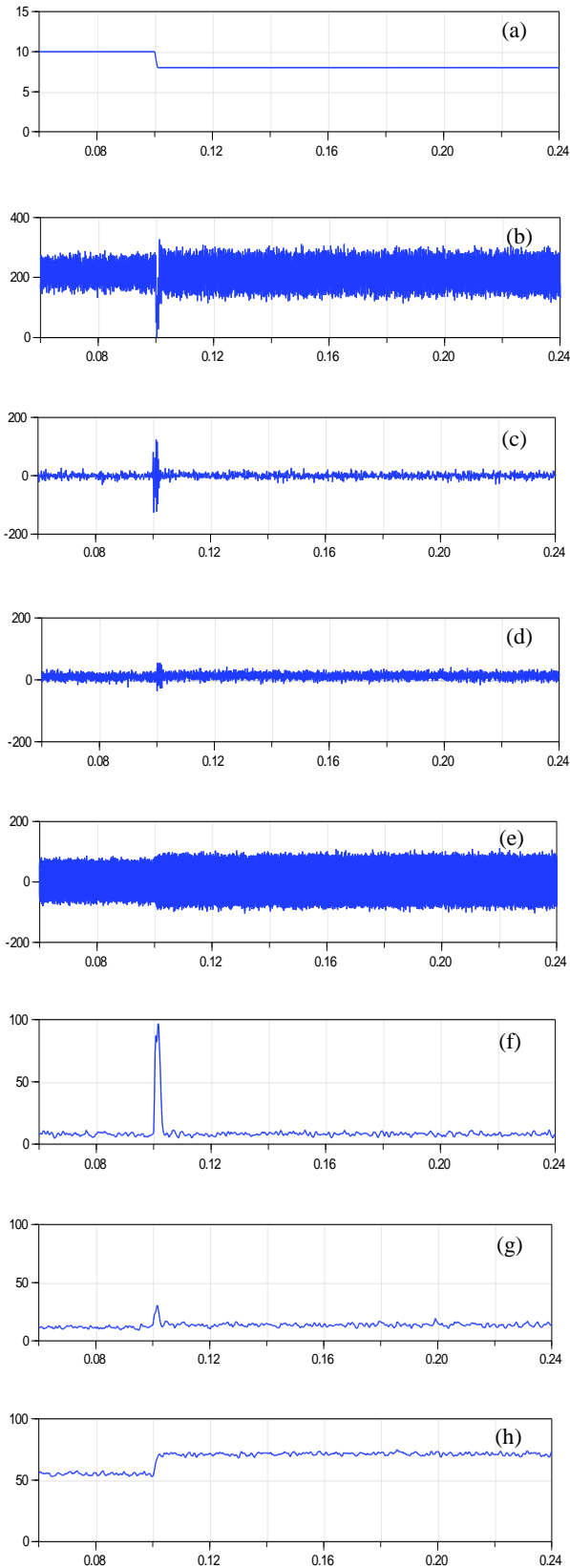


Figure 8: Signals in the detection of a capacitance drop fault within 1 ms using wavelet transform. (a) C_{in} capacitance (μF); (b) input current signal, I_{in} (mA); (c-e) detail wavelet coefficients in MRA from Level 3 to level 1 (mA); (f-h) fault features contained in detail wavelet coefficients of level 3 to level 1 (mA).

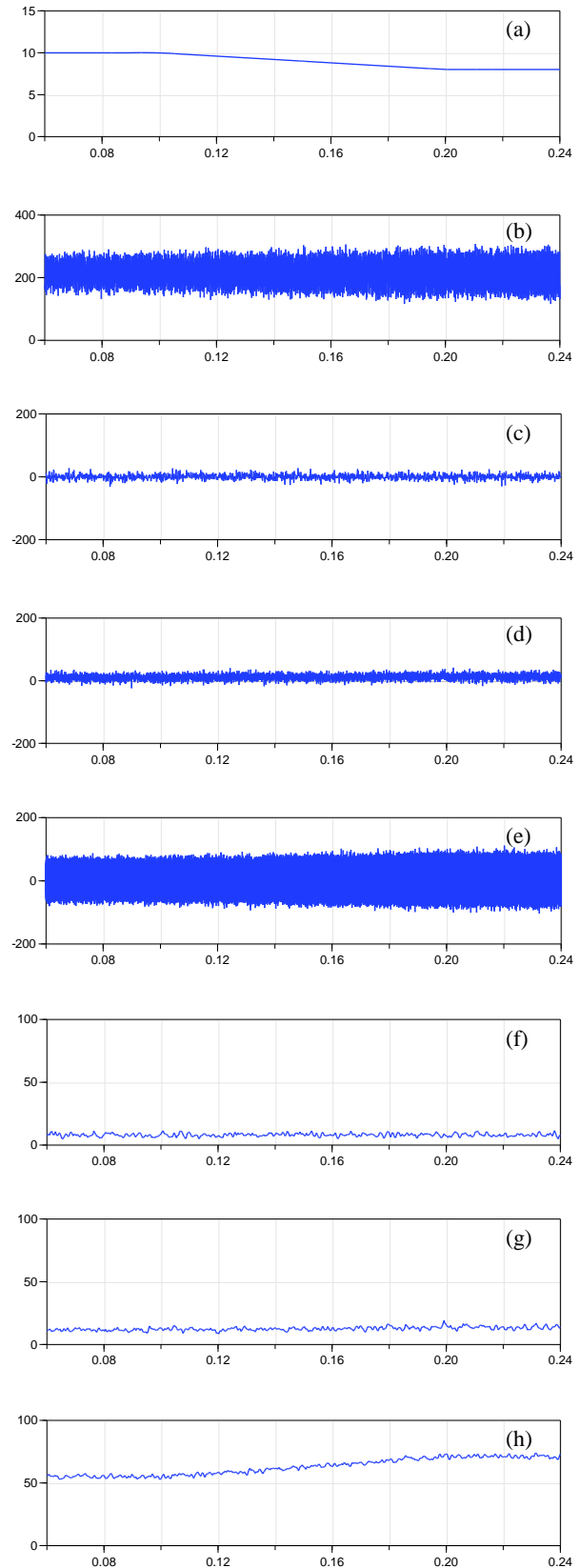


Figure 9: Signals in the detection of a capacitance drop fault within 100 ms using wavelet transform. (a) C_{in} capacitance (μF); (b) input current signal, I_{in} (mA); (c-e) detail wavelet coefficients in MRA from Level 3 to level 1 (mA); (f-h) fault features contained in detail wavelet coefficients of level 3 to level 1 (mA).

4.5 Comparison with filter technique

The wavelet fault detection method was compared with the traditional filter technique, which was processed according to the procedure as shown in Figure 10.

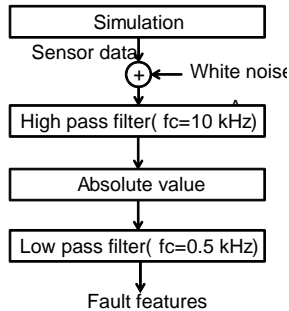


Figure 10: Process for simulation and fault detection using filter technique

The process is similar to that using wavelet transform. The differences are:

- (1) Instead of wavelet decomposition for removing the low frequency normal operation signal, a second order Butterworth high pass filter was used. Its cut off frequency was optimized at $f_c = 10$ kHz to keep as much as possible fault information in high frequency range.
- (2) The output of the high pass filter is a single one dimensional time vector, not as in the wavelet decomposition, where multiple vectors are obtained.

The fault features obtained using both methods for different fault occurrence rates are compared in Figure 11. It shows significant advantages of the wavelet method over the traditional filter method.

For the fast changing faults, the faults were expressed with both linear change of the feature magnitude and much sharper pulses in the wavelet method. In Figure 11a the pulse peak in the wavelet method reached about 80 mA from the base line, while the filter method only showed a peak value of about 10 mA.

For the slow changing faults, shown in Figure 11a and b, although no significant peaks were observed in the higher decomposition levels using wavelet transform, the features in the first level were still clearer than that obtained with the filter method. In the wavelet method, the features showed a difference from 55 to 72.5 mA with a relative change of about 32%, while the filter method gave a difference from 34 to 42.5 mA with a difference of only about 25%.

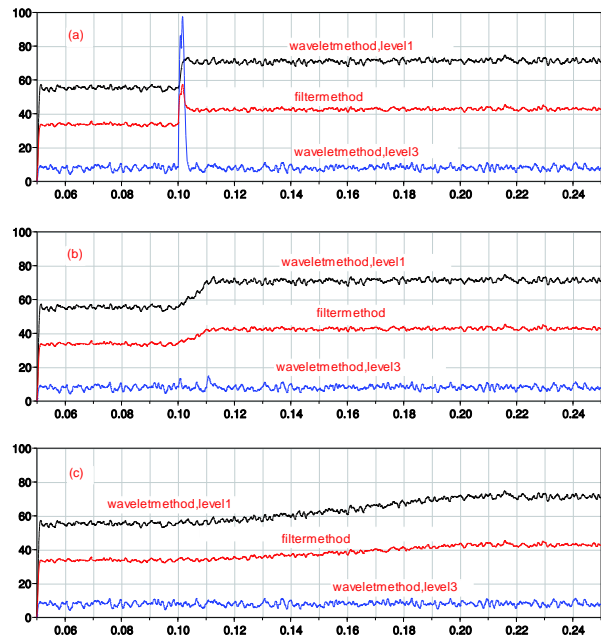


Figure 11: Comparison of the fault features between wavelet method and filter method for (a) 1 ms, (b) 10 ms and (c) 100 ms fault occurrence rate (black solid line --- level 1 wavelet; blue dashed line --- level 3 wavelet; red solid line --- filter method)

5 Conclusion and discussion

This work described a method for early fault detection of important electric components in power supply systems for more electric aircraft (MEA) using wavelet transform. The special properties of wavelet transform suit this method well for changing signal analysis. The simulation study using Modelica under the environment Dymola illustrated its superior feasibility for the detection of fast changing faults, which was significantly better than the traditional filter technique. For the slow changing faults, the wavelet method also gave a significant feature and provided clearer information than the filter technique. Based on these advantages, the wavelet fault detection method is expected to achieve satisfied detection of early faults.

In this study, a specific wavelet library for Modelica is being developed, which possesses the basic functionality of wavelet analysis, including wavelet transform and inverse transform, wavelet decomposition and reconstruction for multi-resolution analysis, and other related functions. The work proved the feasibility of the implementation of wavelet analysis in Modelica.

More work is being done in this topic, including further development of the Modelica wavelet library and experimental study of the fault detection with a

real buck inverter. For a real system, detailed optimization of the fault detection strategy has to be carried out, such as trying other wavelet functions, observing more changing rates, studying the detection with expanded fault ranges, and considering the faults in more electrical parts.

Acknowledgement

The authors thank the support from the Clean-Sky Joint Undertaking through the Grant Agreement No. 296369 (Project MoMoLib) within the Seventh Framework Programme of the European Union.

References

- [1] C. Schallert, A. Pfeiffer and J. Bals. Generator power optimisation for a more-electric aircraft by use of a virtual iron bird, 25th International Congress of the Aeronautical Sciences, 2006.
- [2] M.R. Kuhn, Y. Ji, D. Schröder, Stability studies of critical DC power system component for More Electric Aircraft using μ sensitivity. Proc. of the 15th Mediterranean Conference on Control & Automation, 2007.
- [3] J. Gertler. Fault Detection and Diagnosis in Engineering Systems. Marcel Dekker, 1998.
- [4] J. Chen and P. Patton. Robust Model-based Fault Diagnosis for Dynamic Systems. Kluwer Academic Publishers, 1999.
- [5] S. X. Ding. Model-based Fault Diagnosis Techniques: Design Schemes, Algorithms and Tools. Springer Berlin, 2008.
- [6] K. Swarup and H. Chandrasekharaiah. Fault detection and diagnosis of power systems using artificial neural networks. 1st international forum on application of neural networks to power system, 1991.
- [7] M. Aldeen and F. Crusca. Observer-based fault detection and identification scheme for power systems: Generation, Transmission and Distribution. IEE Proceedings, vol. 153, pp.71-79, 2006.
- [8] Y. Ji and J. Bals. Multi-Model Based Fault Detection for the Power System of More Electric Aircraft. Proceedings of the 7th Asian Control Conference, Hong Kong, China, Aug. 27-29, 2009.
- [9] Y. Ji, J. Bals, Application of model detection techniques to health monitoring for the electrical network of More Electric Aircraft, International Conference on Electrical Engineering and applications, 2009.
- [10] S. Mallat (2009): A wavelet tour of signal processing - the sparse way. Amsterdam: Elsevier.
- [11] H. T. Zhang, Q. An, et al. Fault Detection Wavelet Fractal Method of Circuit of Three-Phase Bridge Rectifier. International Conference on Intelligent System Design and Engineering Application (ISDEA), 2010, S. 725–729.
- [12] V. Prasannamoorthy, N. Devarajan, et al. Wavelet and Fuzzy Classifier Based Fault Detection Methodology for Power Electronic Circuits. International Conference on Process Automation, Control and Computing (PACC), 2011, S. 1–6.
- [13] T. Buente, A. Sahin, and N. Bajcinca, Naim. Inversion of Vehicle Steering Dynamics with Modelica/Dymola. Proceedings of the 4th International Modelica Conference 2005, S. 319–328.
- [14] J. Bals, Y. Ji, M. R. Kuhn, C. Schallert, Model based design and integration of More Electric Aircraft systems using Modelica. Moet forum at European power electronics conference and exhibition, 2009.
- [15] Y. Ji, J. Bals, A Modelica signal analysis tool towards design of More Electric Aircraft, ICIAE, 2010.
- [16] E. Jacobsen and R. Lyons. The sliding DFT. Signal Processing Magazine, vol. 20, issue 2, pp. 74–80, March 2003.
- [17] I. Daubechies. Ten Lectures on Wavelets. SIAM 1992.

PySimulator – A Simulation and Analysis Environment in Python with Plugin Infrastructure

A. Pfeiffer, M. Hellerer, S. Hartweg, M. Otter, M. Reiner
DLR Institute of System Dynamics and Control, Oberpfaffenhofen, Germany
{Andreas.Pfeiffer, Matthias.Hellerer, Stefan.Hartweg, Martin.Otter, Matthias.Reiner}@dlr.de

Abstract

A new simulation and analysis environment in Python is introduced. The environment provides a graphical user interface for simulating different model types (currently Functional Mockup Units and Modelica Models), plotting result variables and applying simulation result analysis tools like Fast Fourier Transform. Additionally advanced tools for linear system analysis are provided that can be applied to the automatically linearized models. The modular concept of the software enables easy development of further plugins for both simulation and analysis.

Keywords: PySimulator; Python; Simulator; FMI; FMU; Modelica; Plugin; Simulation; Analysis; Linear System Analysis

1 Introduction

In this article the open source environment *PySimulator*¹ is introduced and its design is discussed. The central idea is to provide a generic framework

- to perform simulations with different simulation engines in a convenient way,
- organize the persistent storage of results,
- provide plotting and other post-processing feature such as signal processing or linear system analysis, and
- export simulation and analysis results to other environments.

1.1 Design

From an end-user's point of view, PySimulator consists of a convenient graphical user interface so that all these operations can be defined mostly with the mouse. This is similar to many other, usually commercial, simulation environments.

However, the major innovation is that PySimulator is constructed as a *plugin system*: Nearly all operations are defined as plugins with defined interfaces. Several useful plugins are already provided, but anyone can extend this environment by his/her own plugins and there is no formal difference to plugins already provided by the authors of the paper.

Introducing a new plugin means to copy a template and adapt it by writing Python code. Hereby it is possible to build upon the results of other plugins and provide own results to other plugins. All plugin functionality available via the graphical user interface shall also be easily accessible in Python scripts. This will allow a modeler to define and automatically execute Python scripts.

1.2 Related Work

There are several existing Python packages that aim to simulate dynamic systems of standardized physical models like Modelica or FMI [MC10]:

- The software package BuildingsPy [LBN+12] provides functions in Python to start simulations of Modelica models in Dymola [DS12]. Furthermore the result file can be read to process the signals.
- The OMPython package [GFR+12] interfaces the OpenModelica environment with Python. Hence, many functionalities of OpenModelica can be controlled by Python scripts.
- The packages PyFMI and Assimulo [AAF+12] provide Python interfaces for calling functions of a general Functional Mock-Up Unit. Moreover, sophisticated numerical integration algorithms are interfaced or implemented in Assimulo. The user mainly interacts with the packages by Python scripts. A graphical user interface for plotting simulation results is currently available.

These packages concentrate the functionalities on a specific type of model and simulation engine and do not provide a wide range of post-processing features.

¹ PySimulator builds on other Python packages with different license conditions. The most restrictive used is LGPL. Non-GUI functions are under the BSD license.

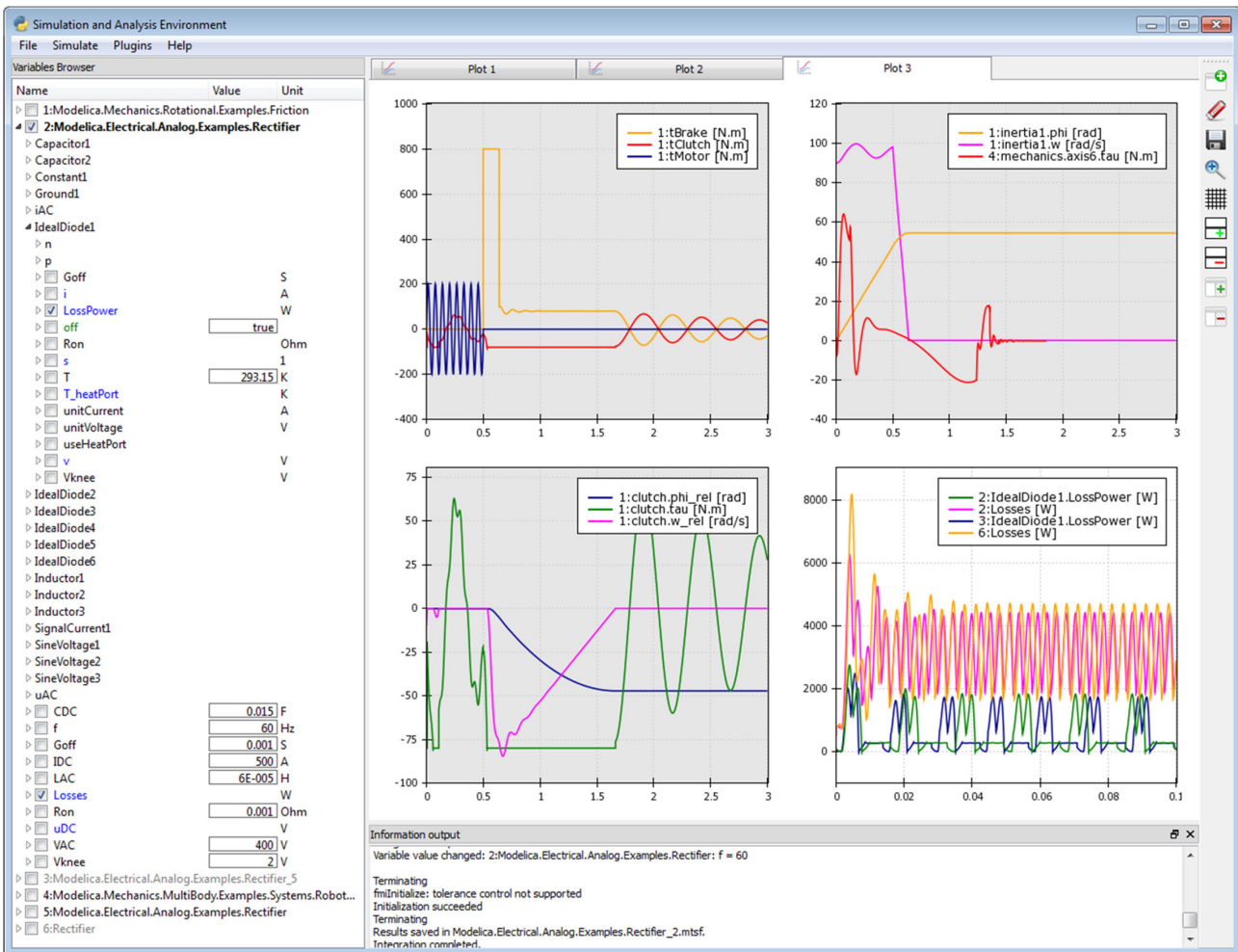


Figure 1: Main Graphical User Interface of PySimulator.

Also, the license conditions are partially restrictive since, e.g., GPL is used. In the Python package index (<http://pypi.python.org>) about 130 Python simulation packages are listed. Most of these packages are dedicated to the simulation of specific models (like neuron networks, biological systems, discrete event systems) or are low level generic packages that require to define a model as Python code (like Assimulo, pyDDE, ScipySim).

2 Architecture and GUI

The environment PySimulator is implemented in Python and depends on several Python packages. The Graphical User Interface (GUI) is built by PySide [P12], a Qt-Interface to Python. Plotting features are realized by integrating Chaco [E12] into the Qt [NC12] framework.

The main GUI of PySimulator (see Figure 1) has a *menu bar* on the top, the *Variable Browser*, a *plotting area* and an *Information output* window. The

menu bar shown in Figure 2 provides functionalities for opening models, opening and conversion of result files, running the simulation and starting analysis tools. In the Variable Browser all models and simulation results are managed to get access to the variables, their attributes and their numeric data. Structured plots show the numeric data in the plotting area.

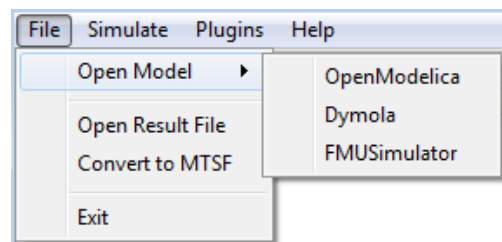


Figure 2: Menu bar of PySimulator GUI.

The environment is intended to provide features for two kinds of users. In a first step the user interactively works with the graphical user interface by loading models, simulating them, plotting variables, applying

analysis tools and inspecting the results. An advanced user can profit from Python’s scripting features because it is possible to load, simulate and analyze models by API function calls in custom scripts.

The implemented software is structured as shown in Figure 3. Some main modules are hosted in the top level directory *PySimulator*. Under *Plugins* all code and data of plugins is organized. Plugin interfaces for model simulators and simulation analysis tools are provided. This plugin concept leads to very modular software that can be easily extended by further plugins. In the following subsections the main GUI elements and the modular plugin structure are presented.

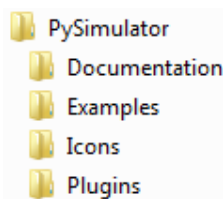


Figure 3: Main directory structure of PySimulator.

2.1 Model and Result Management

A central element of the PySimulator GUI is the *Variable Browser*, see Figure 1. It can show several variable trees of different models. Such a variable tree is either generated by opening a simulation result file, or by opening a model. To open a model the Simulator plugin has firstly to be selected in the menu *Open Model* (see in Figure 2; for more details see Section 3). Secondly the model file itself is to be specified. Each top level item in the Variable Browser

has an ID number followed by a colon and the name of the model. The ID number also marks variables uniquely in plot windows.

By selecting *Open Result File* the user can load a result file of different formats into the Variable Browser. In such a case there is no model to be simulated and the item is displayed in grey color like items number 3 and 6 in Figure 5. Currently two result formats are supported: the proposed standard time series file format MTSF [PBO12] in HDF5, and the binary format generated by Dymola’s [DS12] simulation executable in Matlab 4 format [M12].

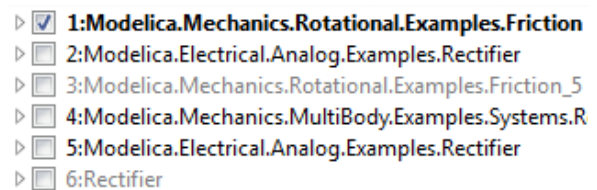


Figure 5: Top level items in the Variable Browser. Black color: Model and result file; grey color: only result file.

For each top level item an information text window (tool tip) is displayed when the user holds the mouse pointer some moment over the name of the item. The text informs about properties of the model. Its structure depends on the model type. An example for an FMU [MC10] is displayed in Figure 4.

After opening a model the variable tree is constructed according to the names of all model variables. New tree branches are introduced by variable names containing dots ‘.’ representing a hierarchy or squared brackets ‘[’, ‘]’ representing arrays. The unit

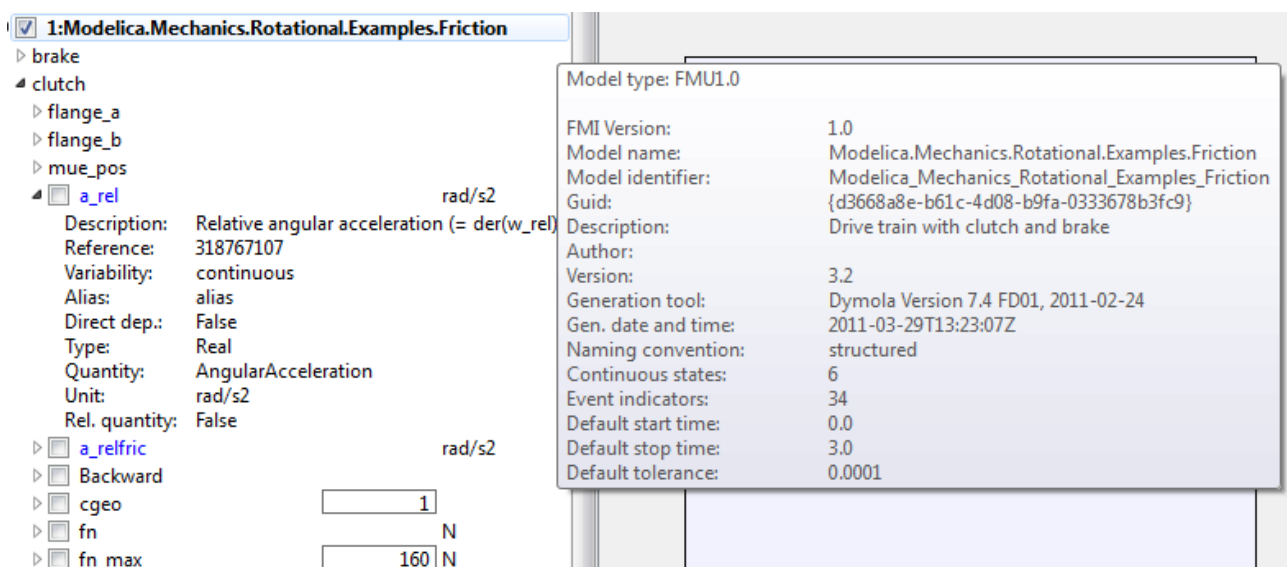


Figure 4: Variable Browser with information text (on the right) for the FMU model.

of the variable is shown if there is any. The values of independent parameters or the initial values of state variables may be edited in the Variable Browser e.g. for `1:clutch.cgeo` or `1:clutch.fn_max`. Furthermore, variables to be plotted can be defined before the numerical integration starts. The attributes of each variable depending on the model or result file type can be displayed by opening the leaf in the variable tree, e.g. for the variable `1:clutch.a_rel` in Figure 4.

During the numerical integration of a model a result file is generated that is associated with the model. A context menu *Results* for the top level items in the Variable Browser informs about the associated result file, see Figure 6. By selecting the context menu *Model* one can close a model and the associated result file. Also, the user can duplicate a loaded model. Each duplicate has its own top level item in the Variable Browser like any other model. It is based on the same model file (e.g. `Friction.fmu` or `Friction.mo`), but has a separate result file, separate settings for the numerical integration and separate values for parameters or initial values set before the numerical integration. For example, the top level item 5 in Figure 5 is a duplicate of model 2 (Rectifier model).

This approach has the advantage that comparing a reference simulation with a tuning simulation of the same original model is very easily possible. The user just duplicates the reference version of the model and experiments on the duplicate. The effects can be directly inspected in plots for the reference and the

tuned version of the model.

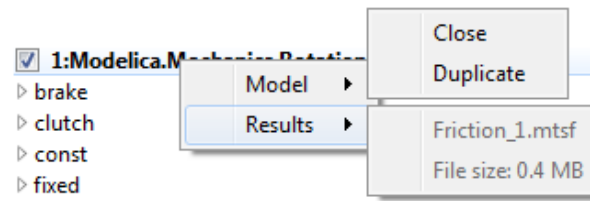


Figure 6: Standard context menus for a top level item in the Variable Browser.

2.2 Plotting Features

Simulation data and analysis results must be processed to make them comprehensible for humans. In PySimulator, the data is visualized using graphical plots. For this purpose PySimulator provides a plotting framework, based on the 2D plotting library *Chaco* [C12]. Chaco was chosen because it is compatible with the Qt/PySide UI-framework, it is licensed under the new BSD license, and it is a native Python library. The last feature not only facilitates the integration but also allows making full use of its object oriented structure at all levels of the inheritance tree for manipulation and extensibility to fit our needs. The primary advantage of Chaco and what sets it apart from other plotting solutions for Python lies in its focus on interactivity.

Plots are shown in a designated area within the main window as shown in the upper right part of Figure 1. Within this area plots can be arranged on a higher level within tabs. The tabs are then subdivided into a

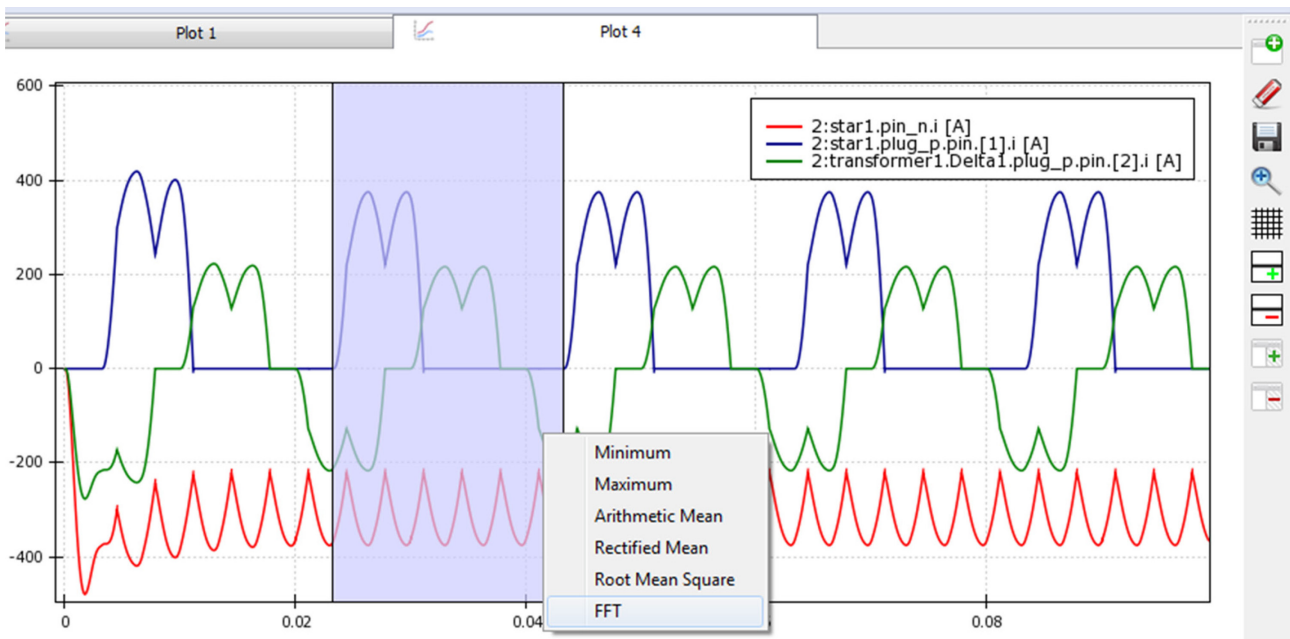


Figure 7: A plot widget displaying three variables, with a selected time interval and open context menu.

grid in which the plots are arranged. To achieve all this, a base class called *PlotWidget* was implemented that acts as an adapter between Chaco and our application or respectively Qt/PySide. All plots are supposed to be implemented as extensions of this base class.

Based on the Chaco framework and *PlotWidget* a default plot widget (*DefaultPlotWidget*) for displaying a variable value over time was implemented while paying special attention to the fact that future plugin developers can both easily use the existing material and still have access to Chaco's full versatility. Marking a variable in the Variable Browser plots the variable value over the time of the simulation in the currently active plot, unmarking it removes the plot line. An example of three variables of a simulated model plotted in a default plot widget can be seen in Figure 7. The following features are based on default Chaco elements and can easily be used individually on any plot, specifically plots by plugins, either out-of-the-box as described here or derived from them to fit special needs:

- *Panning*: Left clicking and dragging within the plot pans the view.
- *Zooming*: Turning the mouse wheel while hovering over the plot zooms in and out. Hovering over an axis only zooms along the respective axis. Zooming and panning works very fast, and is even reasonably fast with millions of points in the plot window.
- *Selecting*: Left clicking and dragging on the X-axis selects a time period. Double clicking the axis opens a menu for textual input of selection limits.
- *Context menu*: Right clicking on the plot opens a context menu. In the *DefaultPlotWidget* it shows callbacks for plugins.
- *Marker*: While hovering over one plot, the plot's time stamp under the mouse is displayed as a vertical line in this and all related plots.

Additionally, plots can be saved as images, either as bitmaps in PNG format or as vector graphics in SVG or PDF format.

2.3 Plugin Structure

Currently, infrastructure for two kinds of plugins is available in *PySimulator*: *Simulator* and *Analysis* plugins. The plugin interfaces are designed to easily integrate own simulator and analysis code.

Simulator plugins are intended to provide the infrastructure to simulate a certain kind of model and write/read the result file of the simulation. In principle all types of simulation engines can be included, provided time series are produced as results and variables and parameters are identified with a hierarchical naming structure. Currently, plugins are available for FMUs [FC10], for Dymola [DS12], and for OpenModelica [GFR+12].

The name of each Simulator plugin appears in the main menu bar (Figure 2) under *Open Model*. To include a Simulator plugin only the plugin code has to be inserted in a new directory of *Plugins/Simulator*, e.g. *FMUSimulator* in Figure 8.

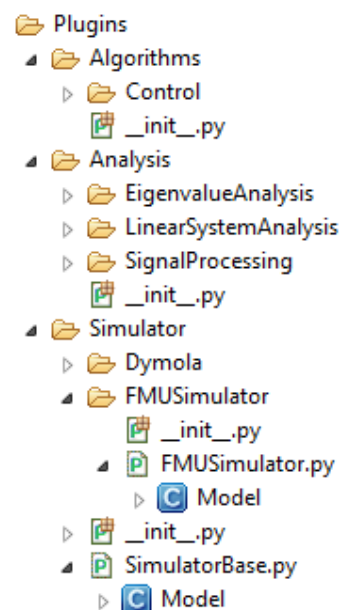


Figure 8: Directory and main Python class structure for plugins in *PySimulator*.

The main Python code of a Simulator plugin has to be inside a class `Model` that is derived from the class `Plugins.Simulator.SimulatorBase.Model`. Important variables, classes and function of the main class `Model` are:

- `modelType`: String, e.g. `'FMUI.0'`, `'Dymola'`.
- `integrationSettings`: Class including start, stop time, algorithm name, etc.
- `integrationStatistics`: Class including number of events, grid points, elapsed real time, etc.
- `integrationResults`: Class including result file access.
- `setVariableTree()`: Function to generate data for a variable tree.

- `getAvailableIntegrationAlgorithms()`: Function to get a list of available integration algorithms.
- `simulate()`: Function to start the numerical integration of the model.
- `initialize(t)`: Function to initialize the model.
- `getDerivatives(t,x)`: Function to evaluate the right hand side of the system.
- `getEventIndicators(t,x)`: Function to evaluate the event indicators (= switching functions to detect events) of the system.
- `getStates()`: Function to get the values of all continuous model states.
- `getStateNames()`: Function to get a list of all names of the continuous model states.
- `getValue(name)`: Function to retrieve the value of a certain variable.

For example, the file *FMUSimulator.py* has a Python class `Model` that provides model typical methods and data as listed for an FMU.

Analysis plugins provide functionality for analyzing the model or result data in the post-processing stage of a simulation. They contain functions which work on variables, models and plots after a model is loaded or a simulation is finished. In order to integrate the Analysis plugins, they are automatically loaded by PySimulator from the *Analysis* folder. An initialization function is called for every plugin to enable the initial setup, like declaration of variables or own classes. The Analysis plugin is further able to register callback functions in the main program which allows access to the plugin's functions. The call of a plugin's function from the GUI takes place by either pull-down menus, a custom button bar or a context menu appearing when the user clicks on an appropriate GUI element like the model's name.

For processing the data, the plugins can implement own algorithms or use shared functionality stored in the *Algorithms* folder. It is furthermore possible for such a plugin to initialize a model or to start a simulation, as this might be necessary for some functionality like linearization of the model. In this case, the features of the Simulator plugins are utilized. The feedback of the Analysis plugin can be sent to the textual Information output window, a plot window or stored in every other way Python allows, e.g. in a file on disk.

It follows a simple example for an Analysis plugin to find the maximum value of a time trajectory and plot a label at the maximum point:

```
def findMax(widget):
    for plot in widget.plots:
        data = plot.data
        maxVal = data[0]
        for time, value in data:
            if value > maxVal[1]:
                maxVal = (time, value)
        maxLabel = DataLabel(
            component=plot,
            data_point=maxVal,
            label_format=str('%(x)f, %(y)f'))
        plot.overlays.append(maxLabel)

def getPlotCallbacks():
    return [{"Find Maximum", findMax}]
```

3 Simulator Plugins

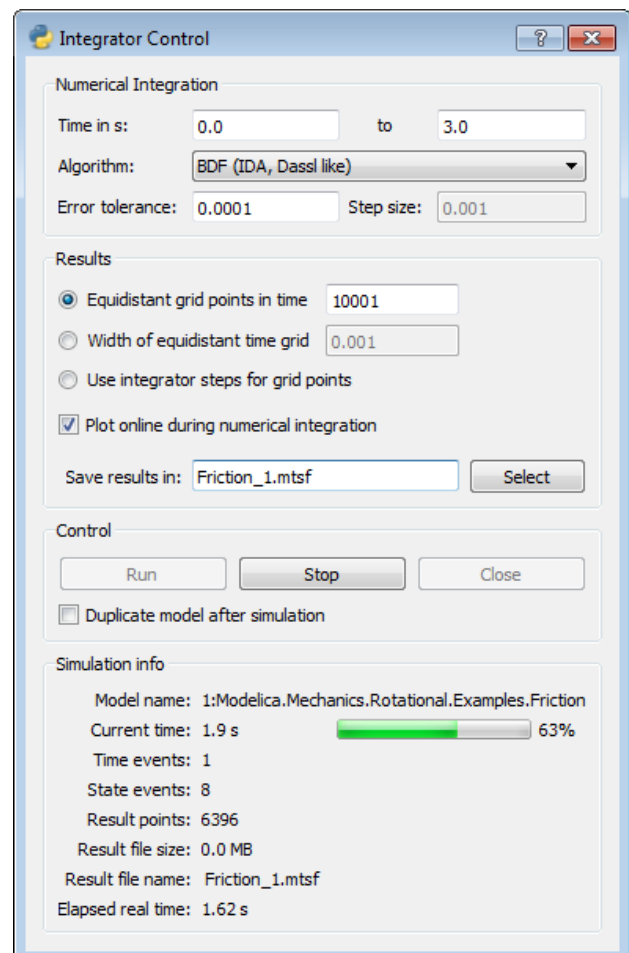


Figure 9: Integrator control GUI in PySimulator.

One of the main features of PySimulator is running and controlling the numerical integration of different types of models (= simulation). Those models require different simulation engines interfaced by the

Simulator plugins in PySimulator. All the Simulator plugins are controlled by the same Integrator Control GUI, see Figure 9. Some menu entries depend on properties of the Simulator plugin.

Start and stop time for the integration may be edited and one of the integration algorithms available for the Simulator plugin can be selected. Depending on the property of the algorithm the user can edit the error tolerance or the fixed step size. The simulation results are mainly discretized, time depending trajectories. The discretization points (= grid points, dense output points) of the time can be given either by the number of equidistant grid points or by the width of an equidistant time grid. A third option is to use the steps of the integration algorithm for the grid points. The name of the result file can also be specified. If *Plot online* is selected in the GUI, the plots of the simulation results are updated during the integration process. This may increase the elapsed real time for the integration, but gives information about the results at once. This feature is especially intended for model simulations that take some time.

The simulation is run in a separate thread, so Variable Browser and Plot area are still available for user interactions. During the numerical integration several statistical parameters inform about the progress: current simulation time, number of time and state events, number of computed result points, the size of the result file and the elapsed real time so far. In some cases it is very helpful to see that for example lots of events are generated and therefore the integration is getting stuck, or the result settings lead to a huge result file and therefore the simulation is slowing down.

3.1 FMU Simulator

The FMU simulator provides an interface to models exported as a Functional Mockup Unit for Model Exchange (FMU, see [MC10]). This interface is supported by more than 30 simulation environments (www.functional-mockup-interface.org/tools.html).

An FMU is basically composed of two components: Firstly, a description file in XML-format holds all information about the variables of the model and other model information. Secondly, binaries for one or several target machines are contained, such as Windows dynamic link libraries (.dll) or Linux shared object libraries (.so). They contain the code for evaluating the model's equations.

This way, the FMU interface allows the evaluation of the right hand side f of the governing equations of a model, as well as its outputs y and its event indicator signals z . They depend in generally on the model's states x , its parameters p , inputs u and the time t . Additionally, time events can be triggered by the FMU. The event indicator signals are used to detect state events, which may occur in many physical models. With this information, it is possible for a numerical integration solver to perform the time integration of the model to obtain a solution, see Figure 10.

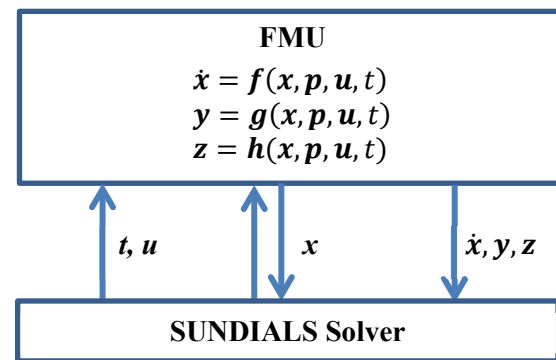


Figure 10: Interface from the FMU model to the SUNDIALS solver.

The single steps performed by PySimulator are the following. First, the XML description file of the selected model is parsed. The information from this file is visualized in the Variable Browser of the main GUI. The Variable Browser can thus also be used independently as an FMU description viewer.

Next, the Functional Mockup Interface (FMI) functions in the shared library are interfaced to make them available in PySimulator. This way, it is possible for the integrator to call the model functions. While these parts are sufficient for some basic operations like initialization, the time integration itself utilizes the Sundials Solver Suite [HBG05]. Sundials provides solvers for explicit and implicit dynamical systems: CVODE and IDA. CVODE numerically integrates ordinary differential equations by linear multistep methods. Depending on the solution CVODE switches between solvers for stiff and non-stiff problems. IDA uses BDF (Backwards Differentiation Formulas) to solve systems of differential-algebraic equations. Sundials supports root finding during the numerical integration. In summary, the Sundials solvers are prepared to be applied to FMUs. The Sundials integrator suite is implemented in C and is accessed from PySimulator via the *python-sundials* [T12] interface.

The FMU Simulator can be interfaced both by code from e.g. an Analysis plugin as well as by the GUI elements described in Section 3. In both cases the important simulation parameters can be adjusted by the user to the specific problem. After simulation, the results are stored in the MTSF file format, the proposed standard time series file format [PBO12] that is based on HDF5 [THG12]. This format offers a way to read and write variable information and numeric data in a convenient and standardized way. The format is especially designed to support both small and very large files. In [PBO12] MTSF files up to 200 GBytes have been generated and variables have been read from the file. Most simulation programs do not support generating and plotting result files of such a size.

For example, a result file for the full robot model from the Modelica Standard Library (FMU generated by Dymola) is generated with 30 Mio. result points. The result file has a size of 171 GBytes. When plotting signals from this file, the loaded signal is downsampled to 5 Mio. points to get acceptable plotting performance.

3.2 Dymola Simulator

The second Simulator plugin is based on the simulation executable (*dymosim[.exe]*) generated by the commercial Modelica environment Dymola [DS12] from Dassault Systèmes. PySimulator supports selecting a Modelica model by asking for the package file and the model name. Then, the Modelica model is automatically compiled by Dymola in the background if there is a version of Dymola installed. The executable includes object code for both the model equations and the numerical integration algorithms.

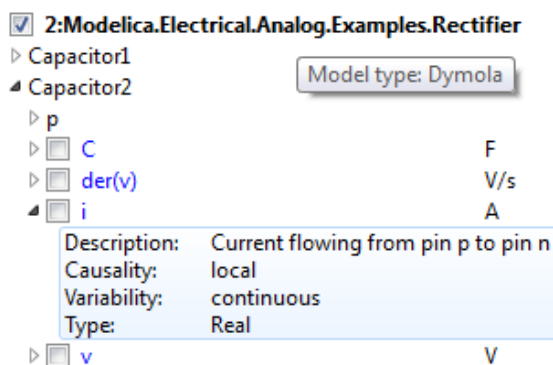


Figure 11: Variable tree in PySimulator based on Dymola's simulation executable.

The list of all variables and the values for editable parameters and initial values are generated when

loading the model, see Figure 11 for an example. If the user wants to start the numerical integration the function `model.simulate` of the Dymola Simulator plugin generates a new initialization file from the integration settings in the Integrator Control GUI and the changed parameters and initial values. After these preparations the simulation executable is started. During the numerical integration process the current simulation time is read and displayed in the Integrator Control GUI to be up to date about the simulation progress.

The result file in Matlab's 4 binary MAT-format can be read by PySimulator. The corresponding result object in PySimulator enables to get access to the numeric data, the description string and the unit by a Modelica variable name. A conversion of Dymola's result file (MAT) to the proposed Standard Time Series File Format (MTSF) is supported by a separate menu entry shown in Figure 2.

3.3 OpenModelica Simulator

A third Simulator plugin for PySimulator is shipped with the open source OpenModelica environment. Details about this plugin are given in [GFR+12].

4 Analysis Plugins

The result of a simulation mainly consists of time series data that can be plotted. Signal processing plugins can access the plot data, can extract more information and can visualize it. Several simple functionalities are already provided to compute minimum, maximum, and other signal properties in a selectable time window. Furthermore, an involved functionality is available to perform Fast Fourier Transformations.

The nonlinear model of a Simulator plugin can be linearized around the initialization point or another time point of the simulation (provided the Simulator plugin supports the required interface for linear models). Afterwards, linear system analysis plugins can operate on such a linear system. Already available plugins compute and plot eigenvalues, provide eigenmode analysis, and perform frequency and step responses.

4.1 Signal Processing Plugin

The Signal Processing plugin provides operations on result signals displayed in a plot window. When right clicking on a plot window, together with an optional

selection of a time range, a window (see Figure 7) pops up to select the desired signal processing operation on the selected time range.

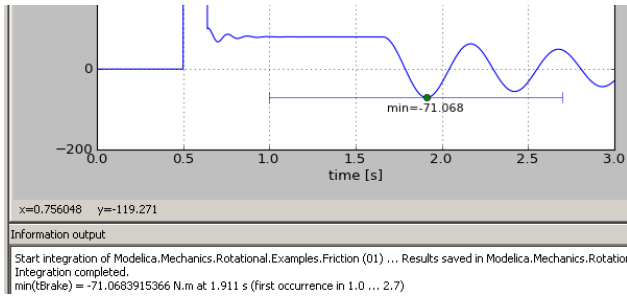


Figure 12: Example for marking of a minimum.

An example of how the result of an operation is shown in a plot is given in Figure 12, where the minimum of a signal is determined in the range $t \in [1.0, 2.7]$.

The operations to be carried out have the following mathematical definition:

Name	Operation on $y(t)$ with $t_{\min} \leq t \leq t_{\max}$, $T = t_{\max} - t_{\min}$
Minimum	$y_{\min} = \min y(t)$
Maximum	$y_{\max} = \max y(t)$
Arithmetic Mean	$y_{DC} = \frac{1}{T} \cdot \int_{t_{\min}}^{t_{\max}} y(t) \cdot dt$
Rectified Mean	$y_{RM} = \frac{1}{T} \cdot \int_{t_{\min}}^{t_{\max}} y(t) \cdot dt$
Root Mean Square	$y_{RMS} = \sqrt{\frac{1}{T} \cdot \int_{t_{\min}}^{t_{\max}} y(t)^2 \cdot dt}$
FFT	$f_s = \frac{n-1}{T},$ $f = \left[0, \frac{f_s}{n}, \frac{2f_s}{n}, \dots, \frac{f_s}{2}\right],$ $\Delta y_r = y(t_r) - y_{DC},$ $y_{FFT,k}(f_k) = \frac{1}{n_f} \sum_{r=0}^{n_f-1} \Delta y_r e^{-i2\pi k \frac{r}{n_f}}$

The integrals in the operations are computed by using the trapezoidal integration rule on the selected signal y (basically, the result points of y are linearly interpolated and then exactly integrated).

The Fast Fourier Transform (FFT, [RKH10]) is used to analyze which frequencies with which amplitudes

are contained in a periodic result signal. For this, a complex vector y_{FFT} is computed as function of a real frequency vector f . Since an FFT requires equidistant time points, the (potentially) non-equidistant result points of a signal, $y = y(t)$, are linearly interpolated and mapped to an equidistant grid of the desired number of points n . The frequency vector f consists of $n_f = \text{div}(n,2) + 1$ points. For even n , the last point of vector f is $f_s/2$, otherwise it is $f_s/2 \cdot (n-1)/n$ (with $f_s = (n-1)/T$ and T as the selected time range). The variant of FFT is used, that subtracts the arithmetic mean of y from the signal y itself and normalizes the FFT result with n_f (in order that amplitudes of y_{FFT} correspond to the amplitudes in the underlying result signal).

The core FFT calculation is performed with Python function `numpy.fft.rfft` which in turn is an interface to the Fortran package `fftpack` [Swa82]. This package computes the FFT of an equidistant vector y of any length n in $O(n^2)$ and if n is expressed as a multiple of 2, 3, 4, or 5, that is $n = 2^i 3^j 4^k 5^l$ in $O(n \cdot \log(n))$ operations. Note, the non-prime factor 4 gives a speed-up with respect to purely 2 factors [Tem83].

A natural question is what number n to select. There are two requirements: (1) all frequencies up to a desired frequency should be included, and (2) the distance between two frequency points should be small enough. With (1) the number of points n can be computed as (T is the time range on which the FFT is applied):

$$f_{\max} = \frac{f_s}{2} = \frac{n-1}{2T} \rightarrow n \approx 2Tf_{\max}.$$

The distance d between two frequency points of vector f for an even number of n is computed as (for an odd n the result is the same, but with a slightly different derivation):

$$d = \frac{f_{\max}}{n_f - 1} = \frac{f_s/2}{n_f - 1} = \frac{\frac{n-1}{2T}}{\frac{n}{2} + 1 - 1} = \frac{1}{T} \frac{n-1}{n} \approx \frac{1}{T}.$$

This means that the frequency resolution depends only on the examined time interval T and can therefore only be enlarged by enlarging this interval (and it is not related to the number of points used in the FFT calculation). For example, if the base frequency is f_0 and the examined time interval T is over k periods of this base frequency, then the distance d is:

$$d = \frac{1}{k/f_0} = \frac{f_0}{k}$$

In other words, in order to get at least a resolution of 10 % of the base frequency, the examined time interval should have at least a range of 10 base periods.

As a simple example consider the following addition of two sines with different amplitudes ($A_1 = 1, A_2 = 0.2$) and frequencies ($f_1 = 5, f_2 = 20$):

$$y(t) = A_1 \sin(2\pi f_1 t) + A_2 \sin(2\pi f_2 t).$$

If 10 periods of f_1 are analyzed, the FFT-plot up to $2f_2$ ($n \approx 2 \cdot \frac{10}{5} \cdot 40 + 1 \rightarrow n = 160$) results in Figure 13.

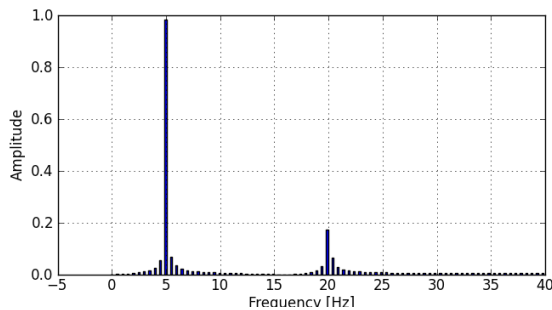


Figure 13: FFT of example with $n = 160$.

As can be seen the 5 and 20 Hz frequencies are correctly identified with small errors in the amplitudes. (the width of the plot bars are selected as $2/5 \cdot d$). Extending the frequency range to $10f_2$ does not change the resolution ($d = 5/10 = 0.5$ Hz), but reduces the amplitude errors as seen in Figure 14.

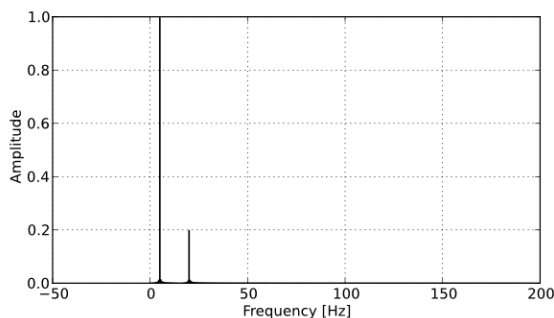


Figure 14: FFT of example with $n = 800$.

4.2 Linear System Analysis Plugin

For many control applications it is necessary to have a linear approximation of a nonlinear system. In addition a linear representation of a nonlinear system

can be helpful to analyze specific properties of the system, for example local stability.

The Linear System Analysis plugin allows to automatically linearize a model that is loaded into PySimulator. If the plugin is loaded, its functionality can be accessed by right-clicking a loaded model in the GUI of PySimulator. If a loaded model is linearized using the GUI the parameter set $p \in \mathbb{R}^{n_p}$, as defined in the Variable Browser is used for the linearization around the operating point. If it is called from a Python-script, a set (Python dictionary) of parameters and values can be used. A model (nonlinear dynamic system) can be represented as a set of equations:

$$\begin{aligned} \dot{x} &= f(x, p, u, t), & x(t_0) &= x_0, \\ y &= g(x, p, u, t). \end{aligned}$$

For the plugin it is necessary that a set of inputs $u \in \mathbb{R}^{n_u}$ and outputs $y \in \mathbb{R}^{n_y}$ are defined in the model, where $n_u \in \mathbb{N}$ is the number of inputs and $n_y \in \mathbb{N}$ is the number of outputs of the system.

The linearization procedure uses a numerical central difference quotient for the calculation of the Jacobians. For a function $q(v)$ depending on a scalar v we use the approximation:

$$q_v(v) \approx \frac{q(v + \delta) - q(v - \delta)}{2\delta}$$

with a step size $\delta = \sqrt[3]{\varepsilon} \max(|v|, 1)$ and the machine precision ε . The step size is computed to find a compromise between a minimum discretization error and a minimum numerical error.

The central difference quotient is successively applied to every component of x and u at a steady state point $w_{ss} := (x_{ss}, p, u_{ss}, t_0)$. The linear approximation of the nonlinear system is a linear time invariant (LTI) system that is represented by the matrices $A \in \mathbb{R}^{n_x \times n_x}$, $B \in \mathbb{R}^{n_x \times n_u}$, $C \in \mathbb{R}^{n_y \times n_x}$ and $D \in \mathbb{R}^{n_y \times n_u}$:

$$\begin{aligned} A &= f_x(w_{ss}), & B &= f_u(w_{ss}), \\ C &= g_x(w_{ss}), & D &= g_u(w_{ss}). \end{aligned}$$

The default case is $x_{ss} := x_0 \in \mathbb{R}^{n_x}$ and $u_{ss} = 0$. If no user defined steady state point is given, x_{ss} is calculated by calling the simulator's initialization function. It is also possible to linearize around an arbitrary user-defined steady state x_{ss} .

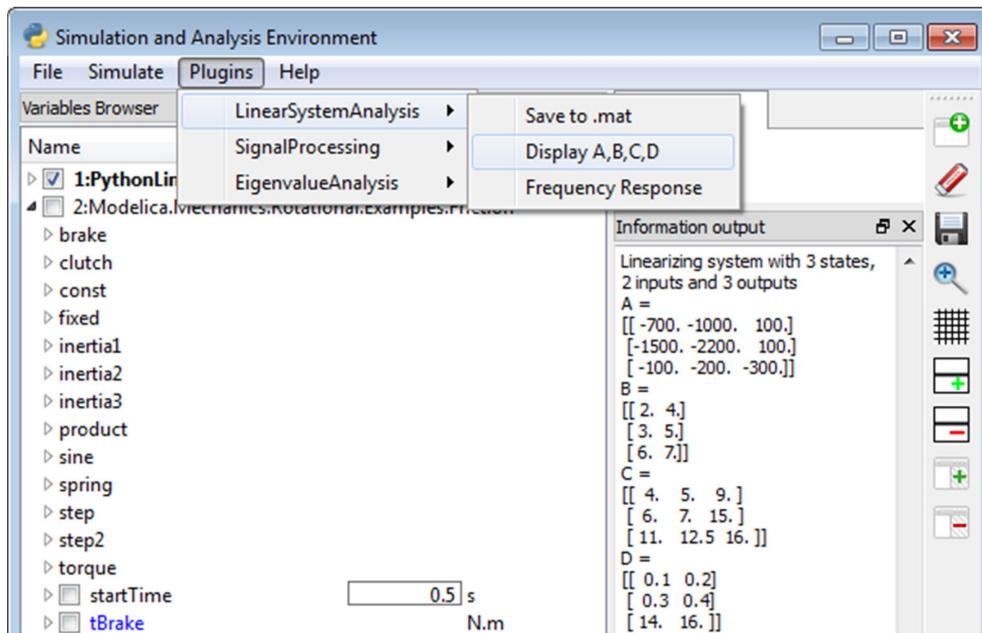


Figure 15: Linear System Analysis plugin inside PySimulator.

The linear system is generated as an instance of a Python class inside the Linear System Analysis plugin, and can be accessed by other plugins inside PySimulator for further analysis. The class provides functions to return the matrices A , B , C , D , names and sizes of the input, output and state vectors. In addition it allows writing the matrices along with the state, input and output names to a file in Matlab's MAT-format, see Figure 15, so that they can be directly used for controller synthesis inside Matlab [M12].

the outputs are computed and plotted. An example of the frequency responses of a system with 2 inputs and 2 outputs is shown in Figure 16.

4.3 Eigenvalue Analysis Plugin

For the analysis of many systems, the eigenvalues and eigenmodes are of special interest. They support the understanding of the system by providing damping and frequency information when eigenmodes or states are excited.

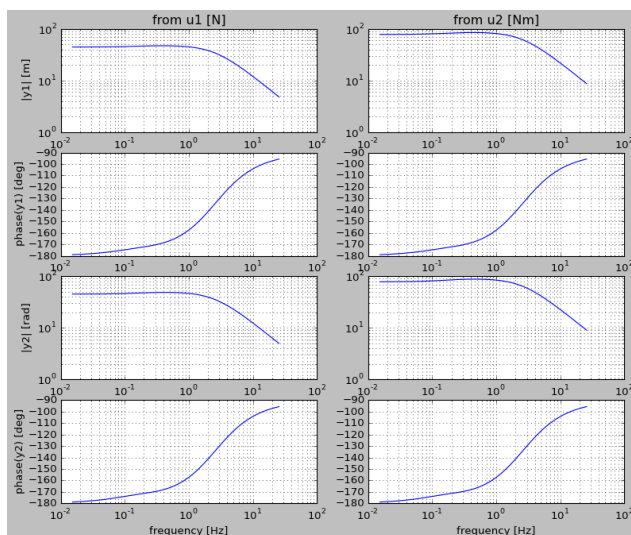


Figure 16: Frequency responses of a 2x2 system.

Furthermore, the plugin provides various analysis operations on the linear input/output system. Most important, the frequency responses from the inputs to

The Eigenvalue Analysis plugin needs the functionality to linearize a system as a starting point for further analysis. For this, the Linear System Analysis plugin from Section 4.2 is utilized. Based on this, functions for the visualization of both eigenvalues and eigenmodes can be called, see Figure 17.

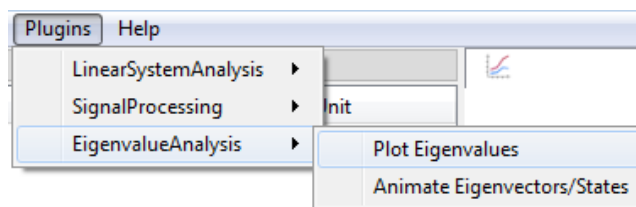


Figure 17: Menu of the Eigenvalue Analysis plugin.

The eigenvalues are plotted in the complex domain as can be seen in Figure 18. This provides information about the stability in the point of linearization as well as about the dynamics of the corresponding eigenmodes. When clicking with the left mouse button on an eigenvalue, additional information to this

eigenvalue is displayed such as frequency, damping and controllability.

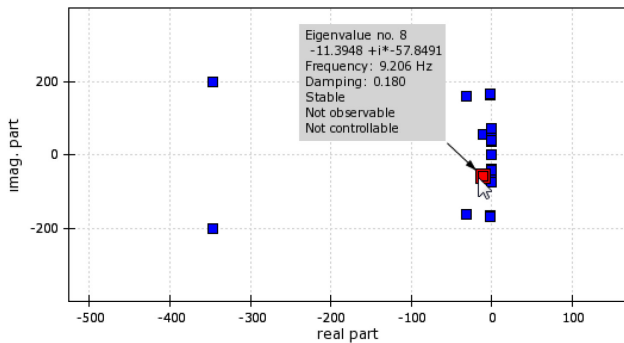


Figure 18: Plot of eigenvalues and frequency response with additional information.

The eigenmodes themselves can be visualized if the model has been exported with an own visualization routine. This is e.g. the case, if a Modelica model is exported with the DLR Visualization library [Bel09]. The eigenmodes are a linear combination of the model’s states. Therefore, they can be visualized if the states have some form of visualization. The selected eigenmodes, see Figure 19, are excited by a periodic sine, making it possible to see their impact on the system, not only in a figure, but in a dynamic way.

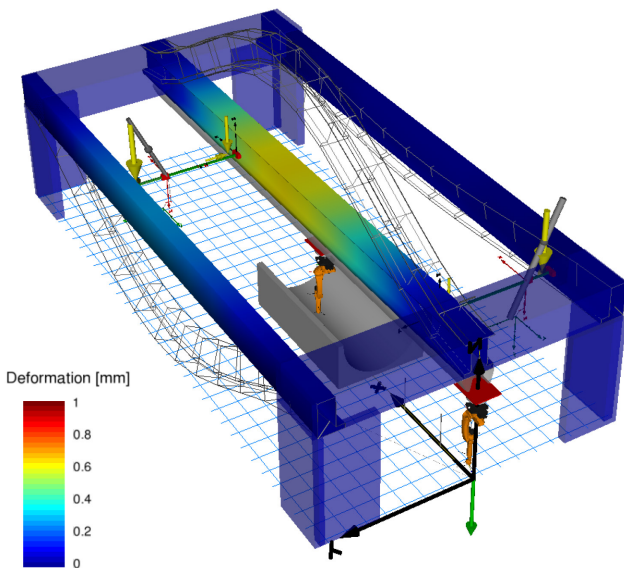


Figure 19: GUI and animation of the 8th eigenmode, showing a clear coupling of the flexible states.

The shown example is a mechanical model of a multi-robot cell of the DLR Center of Lightweight Production Technology. The visualized Eigenmode 8 shows a clear coupling of the left and middle beam due to the portal shown in the upper left part of the

figure. The GUI in Figure 20 shows some dynamic properties which can also be seen in the eigenvalue plot in Figure 18. As an additional possibility, the user can furthermore visualize the states of the system.

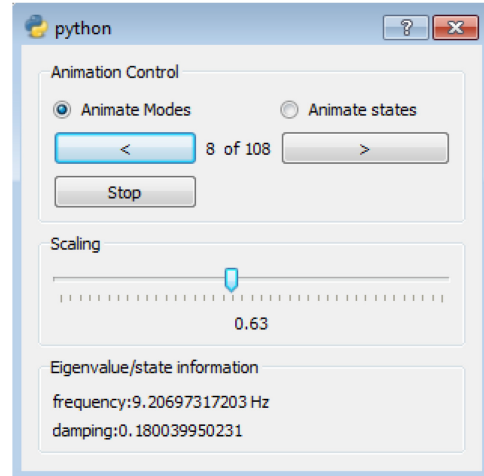


Figure 20: GUI to control the visualization of eigenmodes and states.

The combination of the two abilities *Plot Eigenvalues* and *Animate Eigenvectors/States* enables the engineer to understand and visualize the dynamics of the system. This might help to adapt parameters of the system to e.g. stabilize it or reduce the impact of a periodic disturbance.

5 Algorithms

The algorithms used in the plugins are mostly based on the standard Python packages `numpy` and `scipy`. However, several new algorithms had to be implemented that seemed to be not yet available in other Python packages. These algorithms are provided under directory *Plugins/Algorithms*. All functions in this directory can be used also in any other context, since there is no relationship to PySimulator (just that plugins from PySimulator are calling these functions). Especially, in this directory functions are provided for the Signal Processing and the Linear System Analysis plugins.

For example, class `LTI` in file *Algorithms/Control/lti.py* provides various functions for multi-input-multi-output Linear Time Invariant systems. In the current version, two representations of continuous linear systems are supported:

- **LTI – State Space** (derived by linearization from the nonlinear model, see Section 4.2):

$$\begin{aligned}\dot{x}(t) &= Ax(t) + Bu(t), \\ y(t) &= Cx(t) + Du(t).\end{aligned}$$

- **LTI – Zeros and Poles:**

$$y(s) = \begin{bmatrix} g_{11} & \cdots & g_{1m} \\ \vdots & \ddots & \vdots \\ g_{n1} & \cdots & g_{nm} \end{bmatrix} \cdot u(s),$$

$$\begin{aligned}g_{ij}(s) &= k_{ij} \cdot \frac{\prod_l (s - z_{ij,l})}{\prod_l (s - p_{ij,l})} \\ &= k_{ij} \cdot \frac{\prod_l (s + n_{1,ij,l})}{\prod_l (s + d_{1,ij,l})} \\ &\quad \cdot \frac{\prod_l (s^2 + n_{2,ij,l}s + n_{3,ij,l})}{\prod_l (s^2 + d_{2,ij,l}s + d_{3,ij,l})}\end{aligned}$$

An LTI object is initialized by either defining a state space representation with a tuple of matrices (A , B , C , D), or by defining a zeros and poles representation by a matrix of tuples (k , z , p). Such a tuple is defined with a gain $k \in \mathbb{R}$, and z and p vectors of real or conjugate complex zeros and poles. Internally in the class, a second representation is computed and stored consisting of first and second order transfer functions described by coefficients $n_{q,ij,l}$, $d_{q,ij,l} \in \mathbb{R}$ with $q = 1, 2, 3$. Depending on the selected operation, one of the two representation forms is used to perform the calculation. For example, evaluating a zeros and poles object on a given s -value is performed with the second representation form, since then a real-valued s will result in a real-valued result. Otherwise, due to numerical errors, the result might be complex-valued.

Besides pure data, also meta-information can be associated to an LTI object, consisting of signal names, units and description texts. When generating an LTI object from the Linear System Analysis plugin, this meta information is automatically generated from the corresponding information stored in the underlying model. When plotting or printing an LTI object, the meta-information is utilized to improve the representation for the user.

Currently, only a few operations on LTI objects are provided. Most importantly, a frequency response object can be computed. If the LTI object is in a state space representation, it is internally first transformed to a zeros and poles object and this object is then evaluated on the desired $s = j\omega$ values. By default,

these values are selected on a logarithmic scale and the smallest and largest frequency values are deduced from the poles and zeros. The transformation to zeros and poles form is performed in a numerically reliable way by computing the eigenvalues of A and the generalized eigenvalues of (A, B, C, D) for selected columns of B and selected rows of C and D .

6 Conclusions

PySimulator is provided as an open source environment to conveniently perform simulations with different simulation engines and to analyze the results with a wide range of Analysis plugins. The environment has been designed to cope with large problems. For example, result files with sizes larger than 100 GByte can be handled, as well as several million points in one plot window. We hope that many other people will contribute with Simulator and Analysis plugins. We plan to include plugins from other developers in future PySimulator distributions, provided the plugin adds useful functionality, and the most restrictive license used in the plugin is LGPL. The copyright remains with the developers.

7 Acknowledgement

Important inputs for the design of the Simulator plugin interfaces have been given by Anand Kalaiarasi Ganeson and Peter Fritzson (PELAB) during the fruitful cooperation to integrate the OpenModelica Simulator plugin into PySimulator.

References

- [AAF+12] Andersson C., Andreasson, J., Führer C. and Åkesson J.: *A Workbench for Multibody Systems ODE and DAE Solvers*. In Proc. of 2nd Joint International Conference on Multibody System Dynamics, Stuttgart, Germany, 2012.
- [Bel09] Bellmann T.: *Interactive Simulations and advanced Visualization with Modelica*. Proceedings of 7th International Modelica Conference, Como, Italy, Sep. 20-22, 2009.
- [DS12] Dassault Systèmes AB: *Dymola*, www.dymola.com.
- [E12] Enthougl, Inc.: *Chaco*. code.enthougl.com/chaco.
- [GFR+12] Ganeson A. K., Fritzson P., Rogovchenko O., Asghar A., Sjölund M. and Pfeiffer A.: *An OpenModelica Python Interface and its*

- use in PySimulator*. Accepted for publication in the Proceedings of 9th International Modelica Conference, Munich, Germany, Sept. 2012.
- [HBG05] Hindmarsh A. C., Brown P. N., Grant K. E., Lee S. L., Serban R., Shumaker D. E. and Woodward C. S.: *SUNDIALS: Suite of Non-linear and Differential/Algebraic Equation Solvers*. ACM Transactions on Mathematical Software, 31(3), pp. 363-396, 2005.
- [LBN+12] Lawrence Berkeley National Laboratory: *BuildingsPy*. simulationresearch.lbl.gov/modelica.
- [M12] MathWorks: *Matlab*. www.mathworks.com/products/matlab.
- [MC10] MODELISAR consortium: *Functional Mock-up Interface for Model Exchange, Version 1.0*, 2010. www.functional-mockup-interface.org.
- [NC12] Nokia Corporation: *Qt*. www.qt.nokia.com.
- [P12] PySide. www.pyside.org.
- [PBO12] Pfeiffer A., Bausch-Gall I. and Otter M.: *Proposal for a Standard Time Series File Format in HDF5*. Accepted for publication in the Proceedings of 9th International Modelica Conference, Munich, Germany, Sept. 2012.
- [RKH10] Rao K. R., Kim D. N. and Hwang J.-J.: *Fast Fourier Transform: Algorithms And Applications*. Springer, Dordrecht, Heidelberg, London, 2010.
- [Swa82] Swartztrauber P.N.: *Vectorizing the FFTs*. In: *Parallel Computations*, Ed. G. Rodrigue, Academic Press, 1982, pp. 51-83. www.netlib.org/fftpack
- [T12] Tenfjord R.: *Python-sundials*. www.code.google.com/p/python-sundials.
- [Tem83] Temperton C.: *Self-Sorting Mixed-Radix Fast Fourier Transforms*. Journal of Computational Physics, 52, pp. 1-23, 1983. www.sciencedirect.com/science/article/pii/002199918390013X.
- [THG12] The HDF Group. www.hdfgroup.org.

An OpenModelica Python Interface and its use in PySimulator

Anand Kalaiarasi Ganeson¹, Peter Fritzson¹, Olena Rogovchenko¹, Adeel Asghar¹, Martin Sjölund¹
Andreas Pfeiffer²

¹PELAB – Programming Environment Lab, Dept. Computer Science
Linköping University, SE-581 83 Linköping, Sweden

²Institute of System Dynamics and Control, German Aerospace Center DLR, Oberpfaffenhofen
¹ganan642@student.liu.se, {peter.fritzson, olena.rogovchenko, adeel.asghar, martin.sjолund}@liu.se
²Andreas.Pfeiffer@dlr.de

Abstract

How can Python users be empowered with the robust simulation, compilation and scripting abilities of a non-proprietary object-oriented, equation based modeling language such as Modelica? The immediate objective of this work is to develop an application programming interface for the OpenModelica modeling and simulation environment that would bridge the gap between the two agile programming languages Python and Modelica.

The Python interface to OpenModelica – OMPython, is both a tool and a functional library that allows Python users to realize the full capabilities of OpenModelica's scripting and simulation environment requiring minimal setup actions. OMPython is designed to combine both the simulation and model building processes. Thus domain experts (people writing the models) and computational engineers (people writing the solver code) can work on one unified tool that is industrially viable for optimization of Modelica models, while offering a flexible platform for algorithm development and research.

Keywords: Python, OpenModelica, OMPython, Python, simulation, modeling, Modelica, Python simulator.

1 Introduction

Necessity is the mother of all inventions. Often in science and engineering, the insufficiency of available tools for researchers and developers creates difficulties in exploring and investigating a certain subject. This creates incentives to develop new infrastructures and tools to fill the void. The goal behind the creation of the Python interface to OpenModelica is to create a free, open source, highly portable, Python based interactive session handler for Modelica scripting and modeling, thus catering to the needs of the Python user community.

OMPython – the Python interface to OpenModelica is developed in Python using tool communication based on OmniORB and OmniORBpy - high performance CORBA ORBs for Python. It provides seamless support to the Modelica Standard Library and the Modelica Language Specification [3] supported by OpenModelica [2].

OMPython provides user-friendly features such as:

- Interactive session handling, parsing, interpretation of commands and Modelica expressions for evaluation, simulation, plotting, etc.
- Creating models, using pre-defined models, making component interfaces and annotations.
- Interface to all OpenModelica API calls.
- Optimized result parser that gives access to every element of the OpenModelica Compiler's (OMC) output.
- Helper functions to allow manipulation of nested dictionary data types.
- Easy access to the Modelica Standard library and calling of OpenModelica commands.
- Provides an extensible, deployable and distributable unit for developers.

Since OMPython is designed to function like a library, it can be used from within any Python application that requires the OpenModelica services. OMPython uses the CORBA implementation of OmniORB and OmniORBpy to communicate with the OpenModelica compiler.

2 Using OMPython

This section describes how to use OMPython and also demonstrates its use in creating a simple Modelica model callable from the Python interpreter. It also presents the two modes of operation specifically designed for testing OpenModelica commands and using the OMPython API as a Python library [1].

2.1 Installing OMPython

The two requirements for the operation of the API are installations of OpenModelica 1.8.1 and Python 2.6.

Since OMPython is supplied together with the OpenModelica installer, the standard source distribution of the API can be used to install it to the third party libraries of the installed Python version. Building and installing the module, for example in the Windows systems, is as simple as running one line of command from the terminal.

```
python setup.py install
```

Now OMPython can be imported into any Python application.

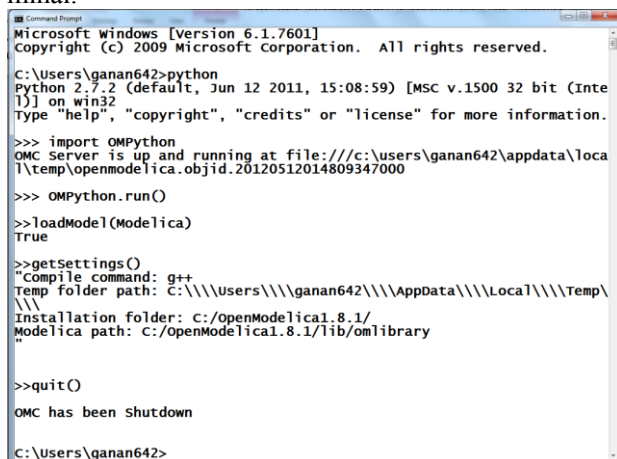
2.2 Executing OMPython

The API can be used in two modes, *Test* and *Library*, each designed for a specific purpose.

2.2.1 Test

Like any new tool, it is important to give its users the freedom to easily explore its capabilities, try its features and possibly suggest new improvements.

For this purpose, the API can be executed in the test mode by executing the `run()` method of the OMPython module. This mode allows users to interactively send OpenModelica commands to OMC via the CORBA interface. The Python types of the OpenModelica output are returned to the user. To illustrate this, in Figure 1 a few operations are presented from the Python terminal.



```
Microsoft windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\ganan642>python
Python 2.7.2 (default, Jun 12 2011, 15:08:59) [MSC v.1500 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.

>>> import OMPython
OMC Server is up and running at file:///c:/users/ganan642/appdata/local/temp/openmodelica.objid.20120512014809347000

>>> OMPython.run()

>>> loadModel(Modelica)
True

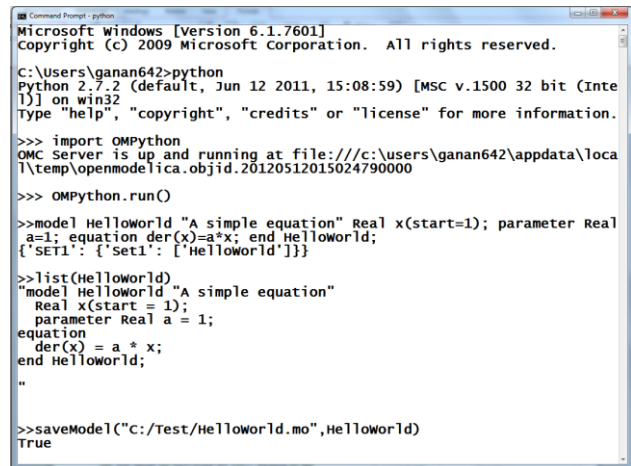
>>> getSettings()
"compile command: g++
temp folder path: C:\\Users\\ganan642\\AppData\\Local\\Temp\\
Installation folder: C:/OpenModelica1.8.1/
Modelica path: C:/OpenModelica1.8.1/lib/omlibrary"

>>> quit()
OMC has been shutdown

C:\Users\ganan642>
```

Figure 1. OMPython executing OpenModelica commands in the Test mode.

Creating new models in the text based Python terminal is rather straightforward using OMPython. Figure 2 illustrates this and shows how a model can be saved with a simple command.



```
Microsoft windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\ganan642>python
Python 2.7.2 (default, Jun 12 2011, 15:08:59) [MSC v.1500 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.

>>> import OMPython
OMC server is up and running at file:///c:/users/ganan642/appdata/local/temp/openmodelica.objid.20120512015024790000

>>> OMPython.run()

>>> model HelloWorld "A simple equation" Real x(start=1); parameter Real a=1; equation der(x)=a*x; end HelloWorld;
{"set1": {"set1": ["HelloWorld"]}}

>>> list(HelloWorld)
"model HelloWorld "A simple equation"
  Real x(start = 1);
  parameter Real a = 1;
  equation
    der(x) = a * x;
  end HelloWorld;
"

>>> saveModel("c:/Test/HelloWorld.mo", HelloWorld)
True
```

Figure 2. Creating and saving a simple HelloWorld model file using OMPython.

2.2.2 Library

Once modelers are familiar with the interface they know what type of responses can be expected and can use the module as a library to programmatically design, simulate, plot, and do more with the models.

This can be done by executing the `execute()` method of the OMPython module. The `execute` method forms the essence of the OMPython API. It encapsulates the OMC operations, CORBA functionalities, parses the results to native Python data types and exposes the API as a simple string processing method. Each instance of the `execute` method returns a result that the modeler can make use of. Additionally, complicated data structures such as deeply nested dictionaries are constructed, strictly typed, and are made available to the user using this method.

The Code Listing 1 shown below provides a simple Python script that uses OMPython as a library to perform a few tasks like loading Modelica libraries to simulating pre-defined Modelica models. Figure 3 depicts the output of the program generated by OMPython on a standard Python terminal.

Code Listing 1

```
import OMPython
OMPython.execute("loadFile(\"c:/OpenModelica1.8.1/testmodels/BouncingBall.mo\")")

result=OMPython.execute("simulate(BouncingBall, stopTime=2, method='Euler')")

print result
OMPython.execute("quit()")
```

```

Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\ganan642>python test_execute_mode.py
OMC Server is up and running at file:///c:/users/ganan642/appdata/local
temp/openmodelica.objid.20120512015521889000

{'simulationOptions': {'options': '***', 'storeInTemp': False, 'cflags':
***, 'simFlags': '***', 'variableFilter': '***', 'noClean': False,
'outputFormat': 'mat', 'method': 'dassl', 'measureTime': False, 's
topTime': 2.0, 'startTime': 0.0, 'numberOfIntervals': 500, 'tolerance'
: 1e-06, 'fileNamePrefix': 'BouncingBall'}, 'simulationResults': {'t
imeCompile': 2.69126048770347, 'timeBackend': 0.0219980482622027, 'mes
sages': '***', 'timeFrontend': 0.0262914864581348, 'timeSimulation': 0.
149678656188106, 'timeTemplates': 0.0174994273747587, 'timesimCode': 0
.00998658420863192, 'timeTotal': 2.9168286378063, 'resultFile': 'c:/u
sers/ganan642/BouncingBall_res.mat'}}

OMC has been shutdown

C:\Users\ganan642>_

```

Figure 3. OMPython executing the Python script shown above.

3 Deploying OMPython in PySimulator

PySimulator is a Python-based Simulation and Analysis tool that is developed by the German Aerospace Center (DLR) in Germany. The tool uses plugins for simulators based on Dymola [10], FMUs [11], and OpenModelica [2]. It also provides analysis tools for some applications particularly in physics and engineering.

This section shows the integration of the new OpenModelica simulator plugin for PySimulator using OMPython.

3.1 The OpenModelica Plugin

The plugin for the OpenModelica simulator integrates easily and well into the PySimulator package by using the OMPython library. PySimulator's template for the plugins provides convenient methods to implement simulation routines, parameter settings, retrieve and use simulation variables and more. Figure 4 shows a part of the development package of PySimulator that includes the OpenModelica plugin.

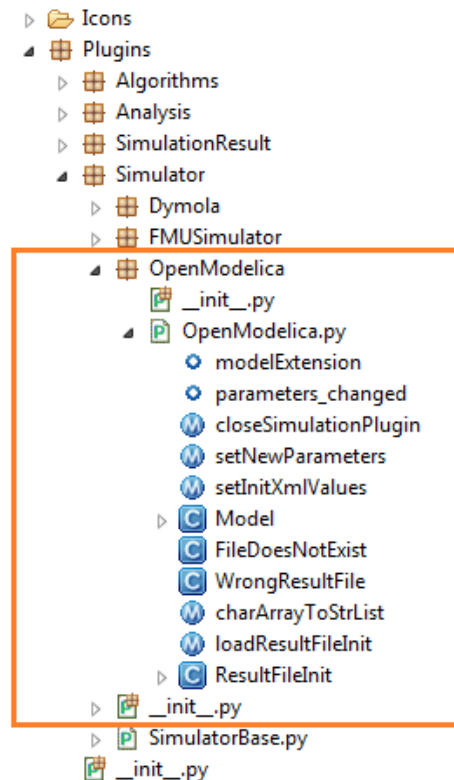


Figure 4. OpenModelica plugin using OMPython within PySimulator.

The OpenModelica plugin defines and uses some features of PySimulator for performing simulations, reading result files, and displaying variables etc. The plugins use PySimulator's plugin templates; this allows other simulation packages to be integrated easily.

The deployment of the OpenModelica plugin within the PySimulator project allows the project to benefit from the full scripting capabilities of the latest OpenModelica API.

3.2 Loading a Modelica Model

The integration of the OMPython module within the OpenModelica plugin for PySimulator makes it possible for the modeler to quickly load Modelica files such as models (.mo) or load a simulated model's executable file.

The user can open these files from the menu bar by selecting `File > Open Model > OpenModelica`.

In this introductory example we will use a pre-defined model named `Influenza` to demonstrate the use of OMPython in PySimulator. Figure 5 depicts the graphical user interface of PySimulator when opening a model file. Once the model file is selected, the model is loaded into the variables browser and is ready to be configured for simulations.

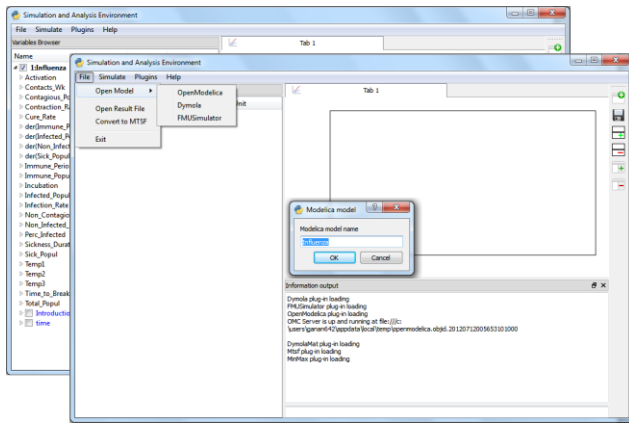


Figure 5. Loading Modelica models or model executables in PySimulator

3.3 Using the OpenModelica plugin

The loaded Modelica model can be simulated from PySimulator using the default simulation options or by setting the simulation options before simulating from the Integrator Control dialog box. The OpenModelica plugin defines the simulation routine for the Modelica models by using the execute method of the OMPython API.

Figure 6 shows how the simulation options can be set using PySimulator's Integrator control feature.

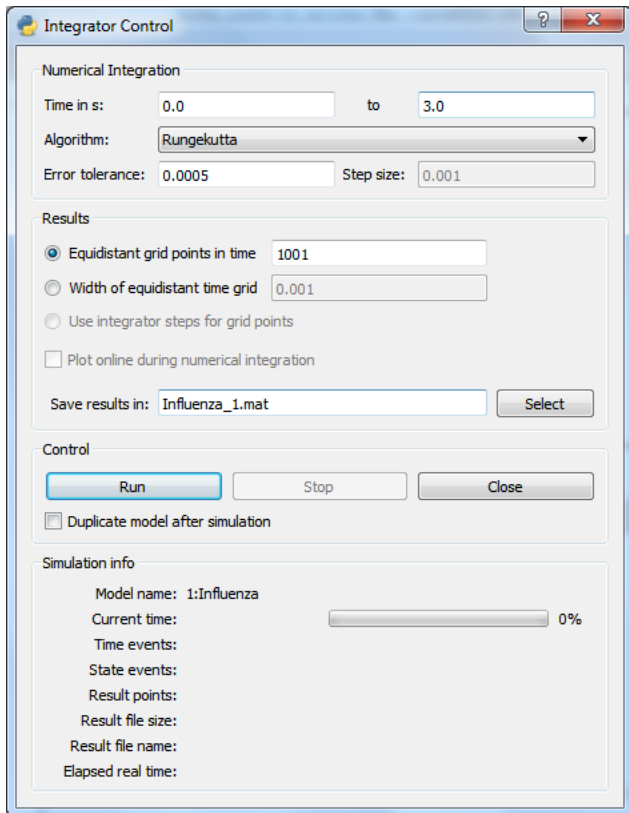


Figure 6. Preparing the simulation settings using the Integrator Control.

3.4 Simulating the model

The initial simulation parameters and settings are provided as inputs to the OMC via the front-end of PySimulator. The Run button of the Integrator control triggers the simulate command of the OMC with the supplied simulation options. The simulate command has the following parameters,

- Simulation Interval
 - Start Time
 - Stop Time
- Algorithm
- Error Tolerance
- Step size

The user has the option to choose from a range of Numerical integration algorithms from the Algorithm selection box. The Integrator control dialog box also filters some parameters that are not available for some integration solvers by disabling the field; avoiding error and providing more accuracy in the results.

The Variables browser builds a tree structure of the instance variables and highlights time-continuous variables in blue. The user can select these variables and plot them in the Plot window by checking the check box near the highlighted variables.

Figure 7 illustrates the Variables browser that allows users to access the variables after the Influenza model has been simulated with some simulation parameters set.

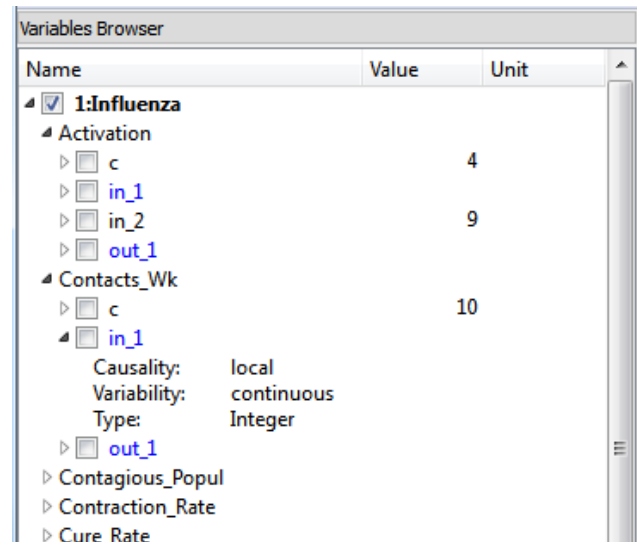


Figure 7. Variables browser of the simulated model.

3.5 Plotting variables from the simulated models

The Plot window of the PySimulator GUI provides additional user interface controls for comparing different

plots side-by-side, adding and removing plots and also to save the plots.

Figure 8 shows the plotted variables in the plot window and the list of simulation variables in the Variables browser along with the variables selected for plotting.

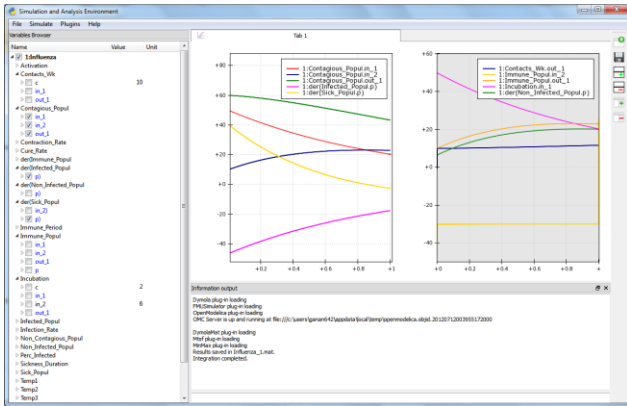


Figure 8. Plotted variables using PySimulator.

3.6 Using Simulated results

It is desirable to avoid simulating the model again every time the user needs the simulation results. It is instead preferable to use an existing simulation result file for the calculations, this saves resources and time. Py-Simulator supports opening the OpenModelica simulation result files (.mat) and the model's executable file to build the variable tree in the variables browser. The user can then adjust some parameters from the variable tree or the Integrator control to achieve the desired results.

4 The OMPython API

The Python interface to OpenModelica addresses its functional requirements through the implementation of two interrelated modules, OMPython and OMParser [1]. This section introduces the two modules and demonstrates their functionalities with some examples.

The following Figure 9 illustrates the functions of the OMPython API with its components.

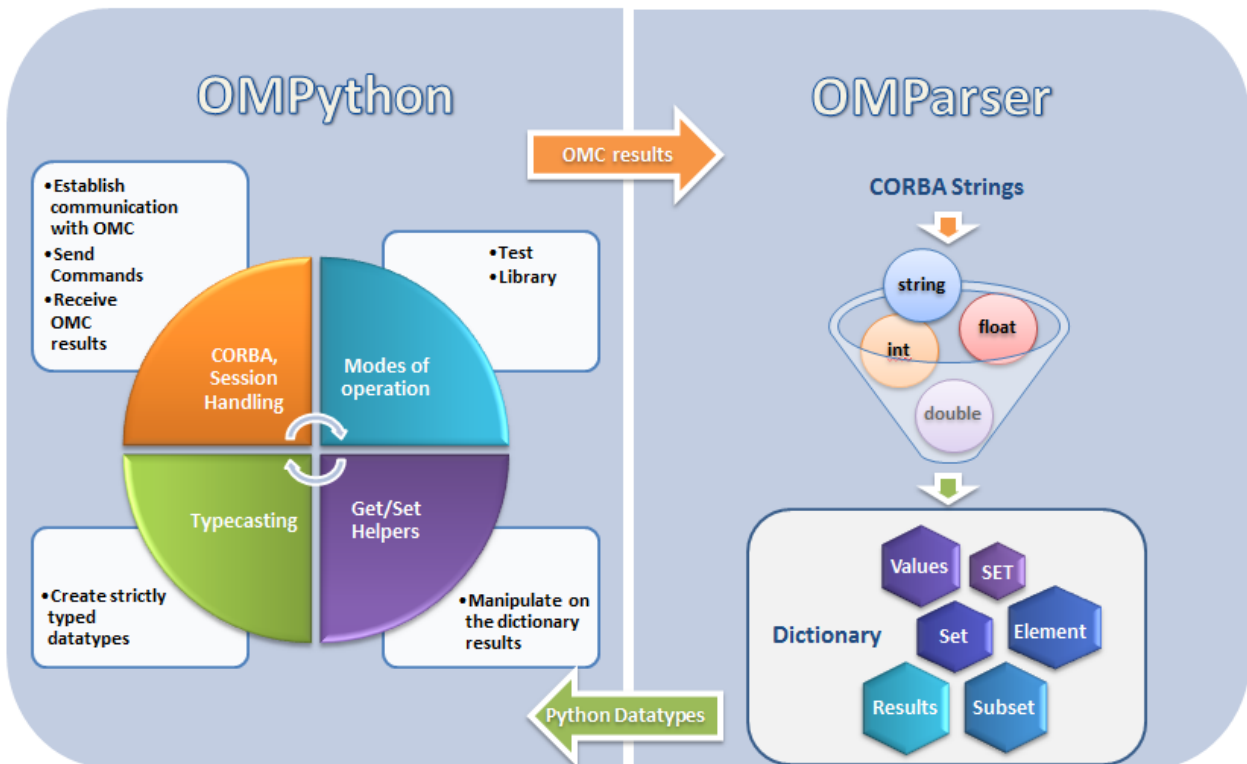


Figure 9. Functions of the OMPython API

4.1 OMPython module

The OMPython module is the main interfacing module of the OMPython API which is responsible for providing the API as a tool and a Python library.

The following are its components:

4.1.1 Interactive Session handler

Each instance of the module creates an interactive session between the user and the OMC. The session handler uses the CORBA Interoperable Object Reference (IOR) file to maintain the user's session activ-

ities and log files for standard output and errors. The session is closed down when the user issues the `quit()` command to the OMC. This also removes the temporary IOR file from the user's machine. The log files facilitate the user with some straight forward debugging and trace-backing purposes.

4.1.2 CORBA Communication

OMPpython uses the Client-Server architecture of the CORBA mechanism to interact with the OMC. OMPython implements the client side of the architecture.

4.1.3 Modes of Operation

The module defines two modes of operation, each designed for specific purposes.

- Test
- Library

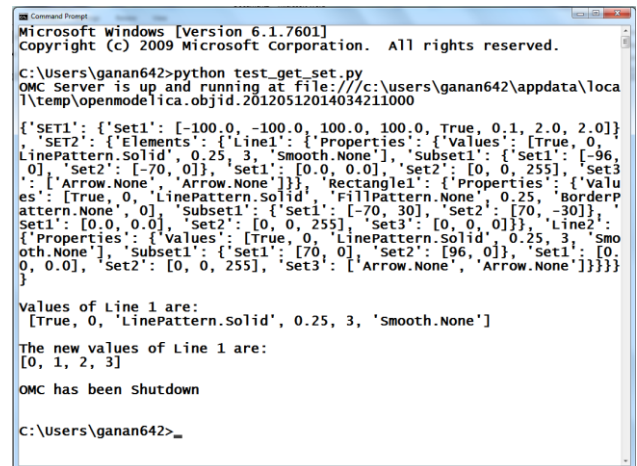
The Test mode allows users to test OMPython while the Library mode gives the user the ability to use the results of OMPython.

4.1.4 Using the interface definition

The vital link between the client and the server processes in this distributed implementation is the Interface Definition Language (IDL) file. OMC defines the `omc_communication.idl` file that it uses to implement the Remote Procedure Calls (RPCs), OMPython mirrors this IDL file to establish the RPC from the client machine.

4.1.5 Get/Set helper functions

Due to the nature of the complicated string outputs generated by the OMC such as Component Annotations, the parser module of the OMPython module generates nested dictionaries. Deeply nested dictionaries in Python require cumbersome operations to retrieve and set values inside dictionaries at various levels. To simplify the multiple steps necessary to perform a get or set operation within a dictionary, OMPython defines the dot-notation `get/set` methods. Figure 10 shows how the user can get and set the values of any nested dictionary data type.



```

Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\ganan642>python test_get_set.py
OMC Server is up and running at file:///c:/Users/ganan642/appdata/local
temp/openmodelica.objid.20120512014034211000

{'SET1': {'Set1': [-100.0, -100.0, 100.0, 100.0, True, 0.1, 2.0, 2.0]},
 'SET2': {'Elements': {'Line1': {'Properties': {'Values': [True, 0,
LinePattern.Solid', 0.25, 3, 'Smooth.None']}, 'Subset1': {'Set1': [-96,
0], 'Set2': [-70, 0]}, 'Set1': [0, 0, 0, 0], 'Set2': [0, 0, 255]}, 'Set3
': ['Arrow.None', 'Arrow.None']}}, 'Rectangle1': {'Properties': {'Valu
es': [True, 0, 'LinePattern.Solid', 'FillPattern.None', 0.25, 'BorderP
attern.None', 0], 'Subset1': {'Set1': [-70, 30], 'Set2': [70, -30]}, '
Set1': [0, 0, 0, 0], 'Set2': [0, 0, 255]}, 'Set3': [0, 0, 0]}, 'Line2':
{'Properties': {'Values': [True, 0, 'LinePattern.Solid', 0.25, 3, 'Smo
oth.None']}, 'Subset1': {'Set1': [70, 0], 'Set2': [96, 0]}, 'Set1': [0,
0, 0, 0], 'Set2': [0, 0, 255]}, 'Set3': ['Arrow.None', 'Arrow.None']}]}}

Values of Line 1 are:
[True, 0, 'LinePattern.Solid', 0.25, 3, 'Smooth.None']

The new values of Line 1 are:
[0, 1, 2, 3]

OMC has been shutdown

C:\Users\ganan642>_

```

Figure 10. Get/Set helper function

4.1.6 Universal Typecaster

Since the variables in Python are dynamically typed, the interpretation of the data types needs to be strictly controlled during runtime. For this purpose, the OMPython module defines a universal typecasting function that typecasts the data to the correct types before building the results.

4.1.7 Imports OMParser

Although the OMC outputs the results to the OMPython module via its CORBA interface, the results are still in the String-to-String CORBA output format which cannot be used intelligibly. So the OMPython module uses its own built-in parser module the OMParser to generate appropriate data structures for the OMC retrieved results.

4.2 OMParser module

Since the results of the OMC are retrieved in a String format over CORBA, some data treatment must be done to ensure that the results are usable correctly in Python.

The OMParser module is designed to do the following,

- Analyze the result string for categorical data.
- Group each category under a category name
- Typecast the data within these categories
- Build suitable data structure to hold these data so that the results are easily accessible.

4.2.1 Understanding the Parsed output

Each command in OpenModelica produces a result that can be categorized according to the statistics of the pattern of data presented in the text. Grammar based parsers were found to be tedious to use be-

cause of the complexity of the patterns of data. This is also the case because the OpenModelica implementation has two types of APIs. One is typed, which could use grammar and the other is untyped, which cannot.

OMPParser follows a few simple rules to parse the OMC output:

- Result strings that do not contain a pair of curly braces "{}" are simply typecasted to their respective types.

For example:

```
>>getVectorizationLimit()
20
>>getNthInheritedClass(Modelica.Electrical.Analog.Basic.Resistor,1)
Modelica.Electrical.Analog.Interfaces.OnePort
```

- Result strings that include one or more pairs of curly braces "{}" are categorized for making dictionary types.

For example:

```
>>getClassNames()
{'SET1':{'Set1': ['ModelicaServices', 'Modelica']}}
```

- Data contained within double quotes "" are formatted to string types; removing the escape sequences in-order to keep the semantics.

For example:

```
>>getModelicaPath()
"C:/OpenModelica1.8.0/lib/omlibrary"
```

4.2.2 The Dictionary data type in Python

Dictionaries are useful as they allow to group data with different data types under one root dictionary name. Dictionaries in Python are indexed by keys unlike sequences, which are indexed by a range of numbers.

It is best to think of dictionaries as an unordered set of *key:value* pairs, with the requirement that the keys are always unique. The common operation on dictionaries is to store a value associate with a key and retrieve the value using the key. This provides us the flexibility of creating keys at runtime and accessing these values using their keys later. All data within the dictionary are stored in a named dictionary. An empty dictionary is represented by a pair of braces {}.

In the result returned by the OMC, the complicated result strings are usually the ones found within the curly braces. In order to make a meaningful categorization of the data within these brackets and to avoid the potential complexities linked to creating dynamic variables, we introduce the following nota-

tions that are used within the dictionaries to categorize the OMC results,

- SET
- Set
- Subset
- Element
- Results
- Values

In this section, to explain these categories, we use the parsed output of OMPython obtained using the Test mode.

4.2.3 SET

A SET (*note the capital letters*) is used to group data that belong to the first set of balanced curly brackets. According to the needed semantics of the results, a SET can contain *Sets, Subsets, Elements, Values and Results*.

A SET can also be empty, denoted by {}. The SETs are named with an increasing index starting from 1 (one). This feature was planned to eliminate the need for dynamic variable creation and having duplicate Keys. The SET belongs within the dictionary called "result".

For example:

```
>>strtok("abcbdef","b")
{'SET1':{'Values': ['"a","c","def"]}}
```

The command `strtok` tokenizes the string "abcbdef" at every occurrence of `b` and produces a SET with values "a", "c", "def". Each value of the SET is then usable in Python.

4.2.4 Set

A set is used to group all data within a SET that is enclosed within a pair of balanced {}s. A Set can contain only Values and Elements. A set can also be empty, it can be depicted as {}, the outer brackets compose the SET, the inner brackets are the Set within the SET.

4.2.5 Subset

A Subset is a two-level deep set that is found within a SET. A subset can contain multiple Sets within its enclosure.

For example:

```
{SET1 {Subset1 {Set1},{Set2},{Set3}}}
```

4.2.6 Element

Elements are the data which are grouped within a pair of Parentheses (). As observed from the OMC result strings, elements have an element name that describes the data within them, so elements can be grouped by their names.

In some cases such as when using the untyped OpenModelica API calls, element structures do not have a name, in these cases the data contained within the parenthesis is parsed into outputs generated by the typed API calls, such as set, values, etc. Also, in some cases many elements have the same names, so they are indexed by increasing numbers starting from 1 (one). Elements have the special property of having one or more Sets and Subsets within them. However, they are still enclosed within the SET.

For example:

```
>>getClassAttributes(test.mymodel)
{'SET1': {'Elements': {'recl':
{'Properties': {'Results': {'comment':
None, 'restriction': 'MODEL',
'startLine': 1, 'partial': False,
'name': "mymodel", 'encapsulated':
False, 'startColumn': 14, 'readonly':
"writable", 'endColumn': 69,
'file': "<interactive>", 'endLine': 1,
'final': False}}}}}}}
```

In this example, the result contains a SET with an Element named `recl` which has Properties which are Results (see section 4.2.7) of the element.

4.2.7 Results

Data that is related by the assignment operator "=", within the SETs are denoted as Results. These assignments cannot be assigned to their actual values unless they are related by a Name = Value relationship. So, they form the sub-dictionary called Results within the Element (for example). These values can then be related and stored using the *key:value* pair relationship.

For example:

```
>>getClassAttributes(test.mymodel)
{'SET1': {'Elements': {'recl':
{'Properties': {'Results': {'comment':
None, 'restriction': 'MODEL',
'startLine': 1, 'partial': False,
'name': "mymodel", 'encapsulated':
False, 'startColumn': 14, 'readonly':
"writable", 'endColumn': 69, 'file':
"<interactive>", 'endLine': 1,
'final': False}}}}}}}
```

4.2.8 Values

Data within any or all of SETs, Sets, Elements and Subsets that are not assignments and separated by commas are grouped together into a list called "Values". The Values list may also contain empty dictionaries, due to Python's representation of a null string "" as {} - an empty dictionary. Although a null string is still a null value, sometimes it is possible to observe data grouped into Values to look like Sets within the Values list.

For example:

```
>>getNthConnection(Modelica.Electrical.Analog.Examples.ChuaCircuit,2)
{'SET1': {'Set1': ['G.n', 'Nr.p', {}]}}
```

4.2.9 The Simulation results

The `simulate()` command produces output that has no SET or Set data in it. Instead, for the sake of simplicity, the result contains two dictionaries namely, SimulationResults and SimulationOptions.

For example:

```
>>simulate(BouncingBall)
{'SimulationOptions': {'options': "",
'storeInTemp': False, 'cflags': "",
'simflags': "", 'variableFilter':
".*'", 'noClean': False,
'outputFormat': "mat", 'method':
"dassl", 'measureTime': False,
'stopTime': 1.0, 'startTime': 0.0,
'numberOfIntervals': 500, 'tolerance':
1e-06, 'fileNamePrefix': "BouncingBall"},
'SimulationResults': {'timeCompile': 4.75231650258347,
'timeBackend': 0.016026309771926,
'messages': None, 'timeFrontend': 1.42004668806536,
'timeSimulation': 0.119703995817784,
'timeTemplates': 0.0230460728977474,
'timeSimCode': 0.0139967955849597,
'timeTotal': 6.3452533928534, 'resultFile': "C:/Users/ganan642/BouncingBall_res.mat"}}
```

4.2.10 The Record types

Some commands produce result strings with Record constructs, these data are categorized for making dictionaries too. To keep the uniformity and simplicity, the data of Record types are grouped into the dictionary `RecordResults`.

For example:

```
>>checkSettings()
{'RecordResults': {'RTLIBS': " -static-libgcc -luuid -lole32 -lws2_32", 'OMC_F
```



```

OUND': True, 'MODELICAUSERCFLAGS': None,
'C_COMPILER_RESPONDING': False, 'OPENMODELICAHOME': '"C:/OpenModelica1.8.1/"',
'CREATE_FILE_WORKS': False, 'SYSTEM_INFO': None, 'CONFIGURE_CMDLINE': '"Manually
created Makefiles for OMDev',
'RecordName':
'OpenModelica.Scripting.CheckSettingsResult', 'OMC_PATH': '"C:/OpenModelica1.8.1//
bin/omc.exe"', 'WORKING_DIRECTORY': '"C:/Users/ganan642"', 'REMOVE_FILE_WORKS':
True, 'OS':
'"Windows_NT"', 'OPENMODELICALIBRARY': '"C:/OpenModelica1.8.1/lib/omlibrary"', 'C_C
OMPILER': '"gcc"'}}

```

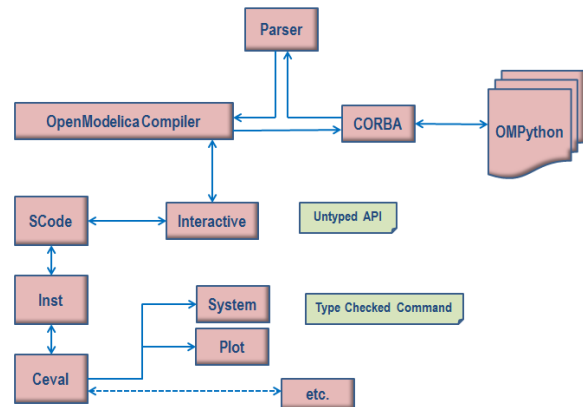


Figure 11. Client-Server of OpenModelica with some interactive tool interfaces

5 OMPython Implementation

The implementation of the OMPython API relies on the Client–Server architecture of CORBA to communicate to the OMC [2]. OMPython acts as the client that requests the services of OMC and OMC behaves like the server and replies to the Python module using the OmniORB and OmniORBpy – Object Request Brokers (ORBs) of CORBA as the communication platform.

This section briefly describes how the API uses CORBA and its other features to achieve its requirements.

5.1 The OMC CORBA interface

The OpenModelica Compiler – OMC can be invoked using two methods:

- Executed at the operating system level, like a program.
- Invoked as a server from a client application using a CORBA client-server interface.

OMPython uses the second method to start OMC since this allows the API to interactively query the compiler/interpreter for its services.

5.2 OMC Client Server architecture

Figure 11 gives an overview of the OpenModelica client server architecture. OMPython plays the role of the client in this architecture. It sends queries and receives replies from the OMC via the CORBA interface. The messages and expressions from the CORBA interface are processed in two groups. The first group consists of the commands which are evaluated by the `Ceval` module and the second group contains the expressions that are handled by the `Interactive` module.

Messages in the CORBA interface are classified into two groups. The first group consists of the user commands or expressions; these are evaluated by the `Ceval` module. The second group contains the declaration of variables, classes, assignments, etc. The client-server API calls are processed by the `Interactive` module.

5.3 Using OMC through CORBA

The OMC process can be invoked from CORBA by executing the OMC executable file using special parameters passed to it. The default location of the OMC executable file is in the `$OPENMODELICAHOME/bin` directory. OMPython invokes OMC with some special flags `+d=interactiveCorba` `+c=random_string` which instructs OMC to start and enable the interactive CORBA communication and also use a timestamp to name the CORBA Interoperable Object Reference (IOR) file that will be created. The timestamp is needed to differentiate the different instances of OMC that have been started by different client processes simultaneously.

The default location where the IOR file is created is in the temp directory. Normally, when OMC is started with the `+d=interactiveCorba` flag, it will create a file named `openmodelica.objid`. On Windows (for example), if the `+c` flag was given, the file name is suffixed with the random string to avoid name conflicts between the simultaneously running OMC server processes. This file contains the CORBA IOR.

5.4 Using the CORBA IOR file

The IOR file contains the CORBA object reference in string format. The CORBA object is created by reading the strings written inside the IOR file.

6 Measurements

In this section, we present some performance measurements for the OMPython API.

The measurements shown are based on the response time of the Python interpreter/compiler that performs the various functions of establishing the CORBA communication, sending commands to the OMC, receiving CORBA outputs, parsing the CORBA outputs and finally displaying the results to the user.

Figure 12 illustrates a simple script that simulates a Modelica model and plots a variable using the Plot generated by OpenModelica. It also shows the received response times of each command that was executed to perform the simulation. Table 1 and Table 2 show the time statistics collected from five unique runs of two simple scripts using the OMPython API. The time is measured in Seconds. Figure 13 and Figure 14 illustrate the overhead between the average output and the unparsed output's response times.

These measurements aim to give an idea about the overhead of the OMPython API in addition to the CORBA overhead that is needed for OMC communication.

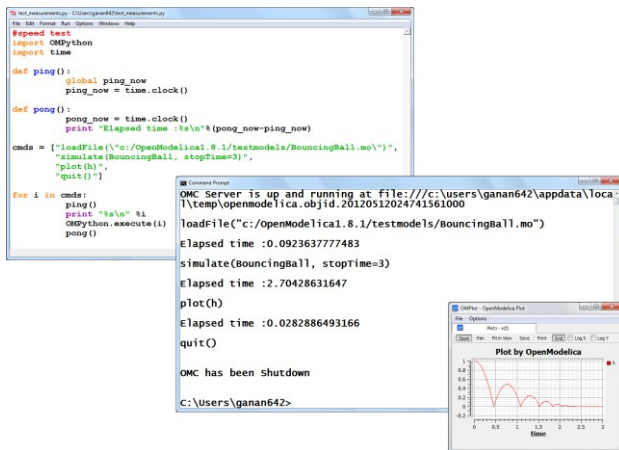


Figure 12. Measuring response times of simulations for the BouncingBall model.

Command	Average response time (s)	Average unparsed response time (s)
loadFile("c:/OpenModelica1.8.1/models/BouncingBall.mo")	0.09223065	0.0421344389

simulate(BouncingBall)	2.60921512	1.8922307169
plot(h)	0.03251472	0.0183359414

Table 1. Response time comparisons for loading, simulating and plotting variables using OMPython.

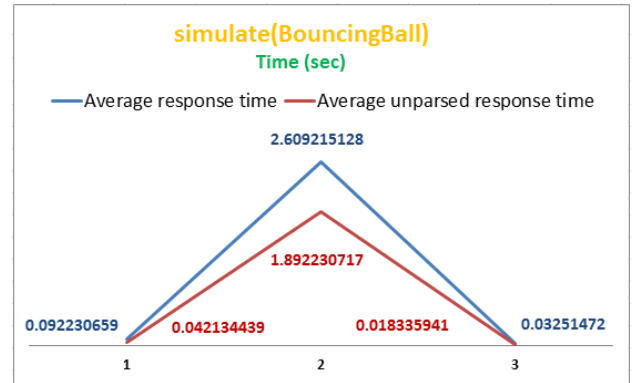


Figure 13. Measuring response time for Simulations in OMPython

Command	Average response time (s)	Average unparsed response time (s)
getVersion()	0.0680588293	0.0590995445
loadModel(Modelica)	5.971103887	4.4708573210
getElementInfo(Modelica.Electrical.Analog.Basic.Resistor)	0.0264064349	0.0190346404
getClassNames()	0.3907942649	0.2707218157
getTempDirectoryPath()	0.0244359882	0.0193691690
getSettings()	0.0327650196	0.0234227783

Table 2. Measuring response times of some OpenModelica commands in OMPython

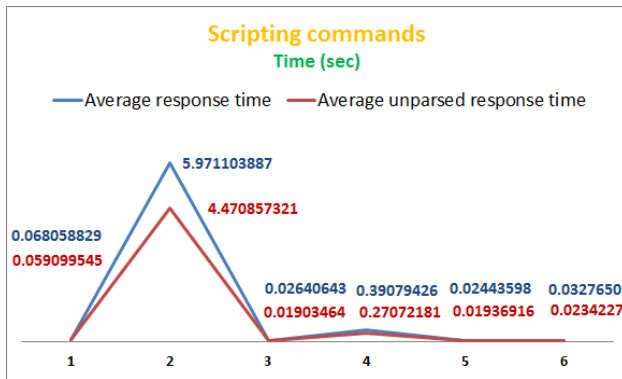


Figure 14. Measuring response times of some OpenModelica commands in OMPython

7 Related Work

Some Simulation packages are available for Python but these packages do not implement an equation-based solving system. Also, they do not provide a Modelica based modeling and simulation environment, but rather present their custom model types.

- *PySCeS* – The Python Simulator for Cellular Systems. It uses the model description language to define its models. Supports solvers like LSODA, sections for non-linear root finding algorithms, Metabolic control analysis, Matplotlib/Gnuplot plotting interfaces, etc. It is released under a new BSD style license and is open source software [4].
- *SimPy* – Simulation in Python, is an object-oriented, process-based discrete-event simulation language for Python. It is released under the GNU Lesser GPL (LGPL) license version 2.1. It features data collection capabilities, GUI and plotting packages. It provides the modeler with the active and passive components of a simulation model and monitor variables for gathering statistics [5]
- *JModelica.org* [12], *MWORKS* [13], and *Amesim* [14] are other Modelica tools providing a Python scripting API.

8 Conclusion

OMPpython is a versatile Python library for OpenModelica that can be used by engineers, scientists, researchers and interested architects to explore and develop Modelica based modeling and simulation efforts. It is free, open source and is distributed with the OpenModelica installation which gives the user the potential to use the full collection of Modelica

libraries that can assist in performing complex simulations and analyses.

The OMPython API places minimal requirements on the user while offering an industry viable standard modeling and simulation environment.

We suggest some future work that can be done to enrich the usage of the OMPython API. The API can be expanded to provide access to the GUI based features of other OpenModelica tools such as OMEdit. User interfaces can be easily built on top of OMPython to implement additional graphic features. Further interesting efforts can be made if the OpenModelica API can be designed to expose its commands as interface definitions in the `omc_communication.idl` file.

9 Acknowledgments

This work has been supported by Serc, by the Swedish Strategic Research Foundation in the EDOp and HIPo projects and Vinnova in the RTSIM and ITEA2 OPENPROD projects. The Open Source Modelica Consortium supports the OpenModelica work.

References

- [1] Anand Kalaiarasi Ganeson. *Design and Implementation of a User Friendly OpenModelica – Python interface*, Master thesis LIU-IDA/LITH-EX-A12/037SE, Linköping University, Sweden, 2012
- [2] Open Source Modelica Consortium. *OpenModelica System Documentation Version 1.8.1*, April 2012. <http://www.openmodelica.org>
- [3] Modelica Association. *The Modelica Language Specification Version 3.2*, March 24th 2010. <http://www.modelica.org>. Modelica Association. *Modelica Standard Library 3.1*. Aug. 2009. <http://www.modelica.org>.
- [4] PySCeS. <http://pysces.sourceforge.net/index.html>
- [5] SimPy. <http://simpy.sourceforge.net/>
- [6] Mark Lutz. *Programming Python*. ISBN 9781449302856, O'Reilly, 2011.
- [7] omniORB 4.1.6 and omniORBpy 3.6. The omni-ORB version 4.1 User's guide, the omniORBpy version 3 User's guide.
- [8] <http://omniorb.sourceforge.net/>
- [9] Andreas Pfeiffer, M. Hellerer, S. Hartweg, Martin Otter, and M. Reiner. *PySimulator – A Simulation and Analysis Environment in Py-*

- thon with Plugin Infrastructure. Submitted to the 9th International Modelica Conference, Munich, Germany, September. 2012.
- [10] Dassault Systèmes AB: la, www.dymola.com.
- [11] MODELISAR consortium: Functional Mock-up Interface for Model Exchange, Version 1.0, 2010. www.functional-mockup-interface.org
- [12] JModelica.org. <http://JModelica.org>. Accessed May 20, 2012.
- [13] MWORKS. <http://en.tongyuan.cc/>. Accessed May 20, 2012.
- [14] LMS Inc. Amesim tool suite. <http://www.lmsintl.com/imagine-amesim-suite>. Accessed May 20, 2012.

WebMWorks: A General Web-Based Modeling and Simulation Environment for Modelica

Liu Qi Xiong Tifan Liu Qinghua Chen Liping

CAD Center, Huazhong University of Science and Technology, Wuhan, China, 430074

luffy.lq@gmail.com xiongtf@hust.edu.cn liuqh@mail.hust.edu.cn chenlp@hustcad.com

Abstract

To meet the requirement of collaboration in the system-level modeling of multi-domain physical systems, a general web-based modeling and simulation environment, WebMWorks, is designed and implemented. It supports multi-user, multi-task and model sharing. Based on MWorks platform, the environment adopts SOA-based architecture and effectively solves the problems of sharing of simulation resources and reuse of the models. By application of RIA technologies, an interactive modeling and simulation environment based on the browser is constructed. This paper introduces the main characteristics and architecture of WebMWorks, and presents the operational effect of the system.

Keywords: visual modeling; web-based simulation; WebMWorks;

1 Introduction

Modelica is a non-proprietary, object-oriented, equation based language, and it has been widely applied because it is conveniently to express the model of complex physical system. Now, Modelica has become one kind of unified modeling standard for multi-domain physical systems. The system-level modeling of multi-domain physical systems using Modelica is difficult to be accomplished by an individual or individual enterprise because of its complexity and multidiscipline. So it is necessary to study the collaborative modeling using Modelica.

The web-based simulation is the integration of the web and simulation technology [1]. Compared to traditional simulation systems, the web-based simulation has many advantages [2], such as, wide usability, cross-platform capability, maintainability, upgradeability, and the sharing of the models.

With the openness, domain-independent and the unified expression of models, Modelica supports the reuse of simulation model based on the model framework at multiple levels. So we can study the

web-based simulation for Modelica and realize the sharing of models and collaborative design in the complex system modeling, which has great practical significance.

At present, the related research is mainly around the virtual experiment and programming languages teaching. In reference [4] a web simulation environment UN-VirtualLab is presented, on which the virtual experiment can be defined by the administrator and the users can modify the experiment parameters to view the results in browser. In reference [5] a web version of the DrModelica is shown. In reference [6] a web-based teaching environment called OMWeb is presented. Student can send their exercises to compile and calculate on the server that contains many OMC (OpenModelica Compiler) wrappers and teachers can view the results. In reference [7] a web-based visual modeling environment was developed for electrical engineering experiment. Users can complete the experiment through connecting the custom experimental components, and the results of the experiment can be obtained after the simulation. However, all the studies presented are based on the specific purpose and lacks some features that are actually needed in industry and research, such as a general web-based visual graphical editor. To overcome these limitations, this paper describes a general web-based modeling and simulation environment: WebMWorks, on which users can easily perform system design, simulation and analysis in the browser. The WebMWorks is based on MWorks [10] and establishes the foundation to form the unified integrated platform for collaborative design and simulation.

2 MWorks Platform

MWorks is a general modeling and simulation platform for complex engineering systems, which provides the compiling and solving engines for WebMWorks.

The framework of MWorks platform is shown in Fig.1.

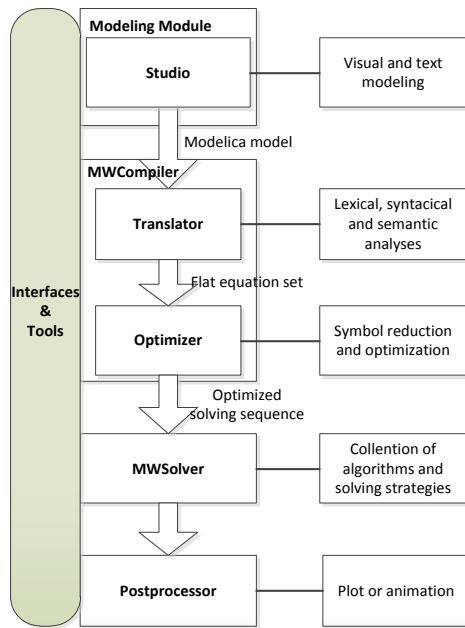


Figure 1: The framework and main process of MWorks

The platform is mainly composed of *Modeling Module*, *MWCompiler*, *MWSolver* and *Postprocessor*. A set of interfaces from MWorks are provided and can be called by external applications. In the WebMWorks, the *MWCompiler* and the *MWSolver* are called to perform the compilation, symbolic reduction and numeric-computation and results can be obtained for the simulation of the Modelica models.

3 Scheme selection

As a prototype platform for collaborative design and simulation based on Modelica, WebMWorks should have the following features:

- 1) Support visual modeling based on web browser.
- 2) Support visualization of simulation results based on web browser.
- 3) Support multi-user and multi-task.
- 4) Support collaborative modeling based on model sharing

In order to realize feature 3) and 4), naturally, Modelica simulator should be located on the server side. Remote compilation and simulation of Modelica models also has been studied and achieved by many researchers [5-9].

Feature 1) is a foundation of the platform. In order to achieve this target, we have two choices: one is downloading Modelica model libraries to the client and analyze the libraries. The users can complete the system modeling with the information of

each model which was obtained by reading its *model text*. Then the system model will be sent to the server and simulated. The other is putting the analysis program for Modelica models on the server side. The client gets the graphical information of each model from the server to realize the visual modeling. Then scenes of the system model are sent to the server. These scenes data will be resolved to *model text*, and finally simulated by calling the compiler.

The first scheme will consume much time when the libraries have a large volume. In this way, it is unable to achieve the advantages of the web system that users can use at anytime, anywhere. And it is also not conducive to protect the intellectual property, when the system contains some private model libraries. The second scheme needs to maintain a free communication between the client and the server in the process of modeling. In spite of this, in order to explore the full benefits of web-based simulation, the second scheme should be chosen.

4 System Design

4.1 System Architecture

To build a modeling and simulation architecture that have indifferent interfaces, is reusable and loosely-coupled, WebMWorks adopts the idea of *Service Oriented Architecture* (SOA) which is implemented by using WCF technology. WCF is a unified programming model provided by Microsoft for building service-oriented applications. With WCF, the core functions of the system is wrapped as a service, and called by the internal or external program. The system based on a layered architecture is shown in Fig.2.

- *Presentation tier*

The presentation tier is a web portal which contains a Silverlight plug-in. It can be used to provide visual modeling and the visualization of the simulation results. Silverlight is a RIA (Rich Internet Application) technology released by Microsoft, with the capabilities of cross-browser and cross-platform.

- *Service tier*

This tier consists of three independent servers, including the web server, the modeling server and the simulation server. By using WCF technology to package the functions of the program on the server, *the presentation tier* can get the variety of services from *the service tier*.

Web server is used to host the client portal, containing the web pages and silverlight plug-

in package. It handles various requests from the client, and provides the services of user management and model management.

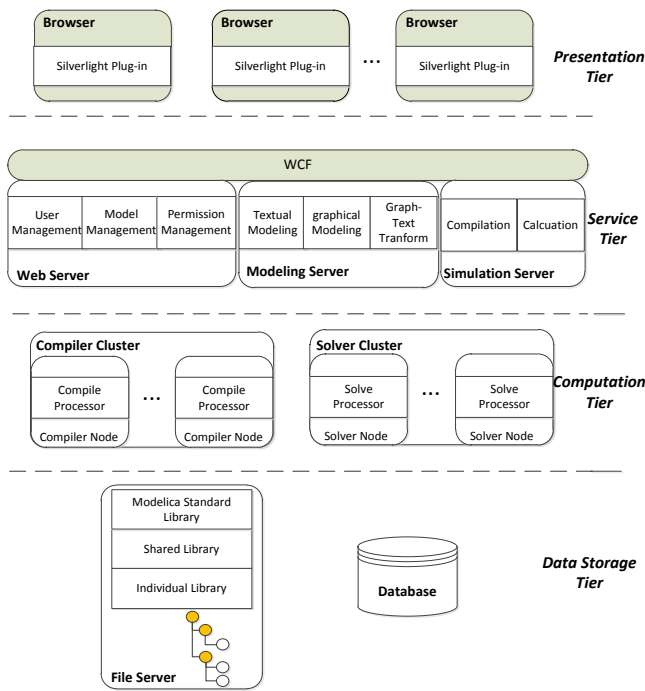


Figure 2: System architecture of WebMWorks

Modeling server receives requests from the user while modeling, including textual modeling request and graphical modeling request. It provides the services of textual modeling, graphical modeling and graph-text transformation.

Simulation server receives requests of compilation and solution from the client. It provides the compilation service and computation service for the *presentation tier*. There are two message queues in this server, the compilation queue and the solution queue. These queues are realized using Microsoft Message Queuing which has many advantages: stability, priority of the message, security and so on. When a compilation or solution request is received, simulation server will put it into the corresponding queue. Then the server makes a balanced distribution of the requests to the nodes of compiler cluster or solver cluster.

- *Computation tier*

The tier consists of two computing cluster: *Compiler cluster* and *Solver cluster*.

Each node in *Compiler cluster* performs the same function, and there is one or more process programs called *CompileProcessor* for processing compilation tasks in it. The number

of processors is determined by the service application on the simulation server. The workflow in the node is shown in Fig.3.

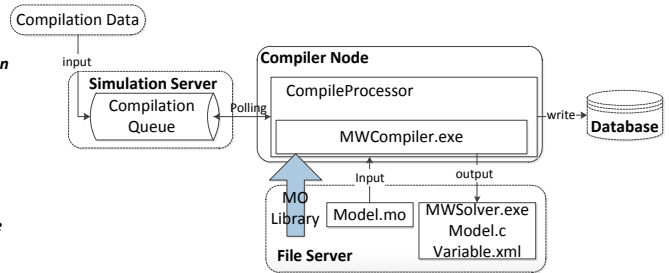


Figure 3: Workflow of each Compiler Node

When a compilation command is executed at the client-side, a request contains compilation data in XML format will be sent into the compilation queue on the simulation server. Compilation data contains several properties of the model to be compiled, including owner, ID, path and so on. *CompileProcessor* is a wrapper for *MWCompiler* which is the compiler of MWorks. It is constantly polling the compilation queue to get the compilation data. Then it calls the *MWCompiler* to load the MSL, the Shared library and the active user's library from the file server. The compiler takes the mo text of one model as input, and outputs the corresponding solver (*MWsolver.exe*), C code, and a XML file for the description of variables. Also, the compilation information will be stored to the database.

Similar to the *Compiler cluster*, *Solver cluster* is composed of several solver nodes which distribute the solution request. Each node can start one or more *SolveProcessor* to process the solution tasks. The workflow is shown in Fig.4.

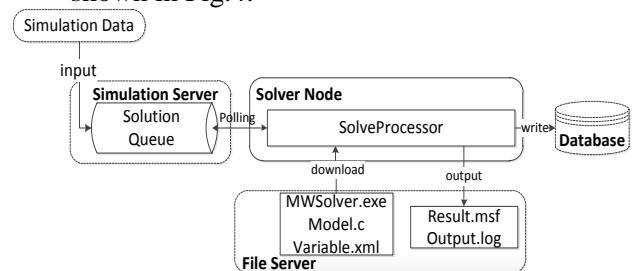


Figure 4: Workflow of each Solver Node

When a simulation command is executed at the client-side, a request contains simulation data in XML format will be sent into the solution queue on the simulation server. Simulation data contains three parts of information: the first is the model information, including owner, ID, path and so on; the second is the setting information of simulation, including start/stop time,

step length, algorithm; the third is the simulation parameters which refer to the modified parameters by users in the post-processing rather than the initial parameters. *SolveProcessor* will be constantly polling the solution queue to get the simulation data and then the *MWSolver* will be downloaded from the file server. *MWSolver* takes simulation data as input, and outputs the result file and the log file of the solution.

- *Data storage tier*

This tier consists of the database server and file server. Database server manages the information of the users and the metadata of Modelica models. The metadata contains the relationship between models and the properties of the Modelica models, such as type, name, path, description and owner. File server is used to store mo files of the models following the storage rules of Modelica models. It also stores SVG files, result files of the simulation, and kinds of intermediate files.

4.2 System Workflow

Fig.5 shows the typical procedure, from logging in the system, to creating a new system model, and finally upto gaining the simulation results.

- 1) *System initialization.* When the modeling sever is up, the modeling application loads Modelica Standard Library (MSL) and Shared library from the file server, and provides all needed services in the process of modeling.
- 2) *User environment initialization.* After successful login, the user will be assigned the appropriate permissions. Simultaneously the modeling application loads individual library of the user from the file server to the memory in.
- 3) *System graphical modeling.* In the modeling page, the user can create a system model in interactive environment, as in the traditional Modelica IDE, such as Dymona, MWorks or OMEdit. The information which required in the modeling like the icons, parameters, properties of the components are achieved through calling the services provided by the *service tier*.
- 4) *Model compilation.* First, the client submits the scene presentation of one system model in XML format to the modeling application and calls the

graph-text transformation service provided by the application. The scene will be resolved to *mo text* in the modeling server and the *mo text* will be saved to the file server. Second, the client calls the compilation services to send the compilation request into the compilation queue. Finally, the *CompilerProcessor* gets the compilation data from the queue, compiles the mo file of the system model and generates the corresponding solver.

- 5) *Model solution.* The request of the simulation from the client will be put into the solution queue. Then the *SolveProcessor* in one solver node gets the simulation data from the queue and calls the *MWSolver* of the model to generate the result data of simulation.
- 6) *Post-proessing.* In the post-processing page, the user can monitor the process of simulation. When the solution is completed, the result data can be packed into XML and sent to the client and displayed on the page.

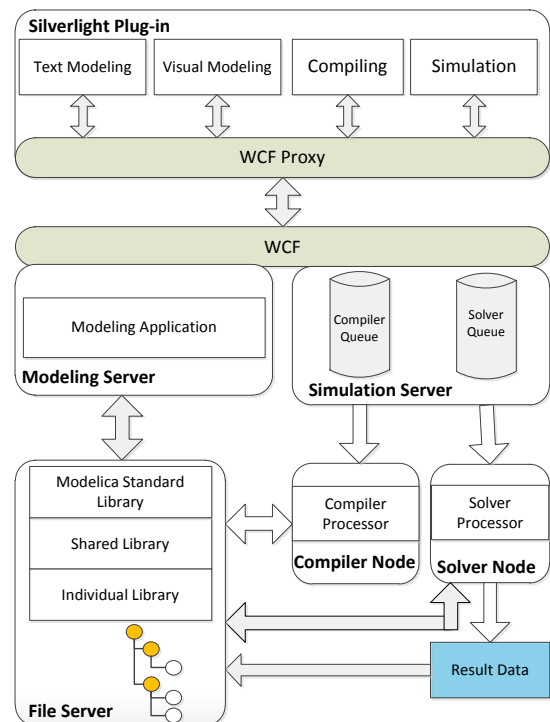


Figure 5: System workflow of modeling and simulation

5 System Implementation

5.1 Client: Modeling Page

The client is developed in a MVVM [12] architecture using the Silverlight technology. The vector graphics and asynchronous communication of Silverlight make it easy to create interactive graphical application in the browser. While a large number of graphical operations are transferred from the server to the client, the burden of the server is lightened, which allows the server with same hardware to handle more requests.

In the modeling page, a visual modeling environment, which is similar with MWork Studio, has been realized. It supports users to create, modify, delete, query and download models. Each model has three views: text view, icon view and diagram view. The screenshot of the modeling page is shown in Fig.6.

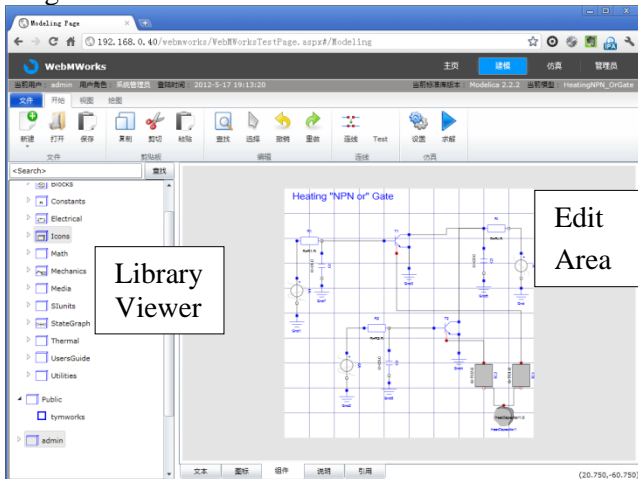


Figure 6: Modeling Environment in Browser

The library viewer contains three root nodes: “Modelica”, “Public” and “CurrentUserName”. The “Modelica” node represents a specific version of MSL. The user can decide which version of MSL to load in the login page. The “Public” node represents the models that the users shared. And the “CurrentUserName” Node is on behalf of the models owned by the user who has logged in. The models under the “Public” and “CurrentUserName” node are consistent with the version of the MSL. For example, when the user selects the version 2.2.2 of MSL, they have only loaded the models based on the Modelica Standard Library 2.2.2. The tree of the library viewer adopts the lazy loading mode, that is, only when the user expands one node, the next level nodes of the node are loaded.

The edit area acts as visual modeling, text modeling or icon-editing area. The icon view provides basic graphics drawing, and the user can edit the icon

of the model in this view. In the Diagram view, the model can be dragged and dropped from the library viewer to this view, and then the model will be instantiated to a component. The users can complete the system modeling by connecting components.

5.2 Client: Post-processing Page

In the Post-processing page, the visualization of simulation results, which is similar with the MWorks Simulator, has been realized. It offers simulation management, setting up and viewing the results of several simulation cases at the same time. The screenshot of the Post-processing page is shown in Fig.7:

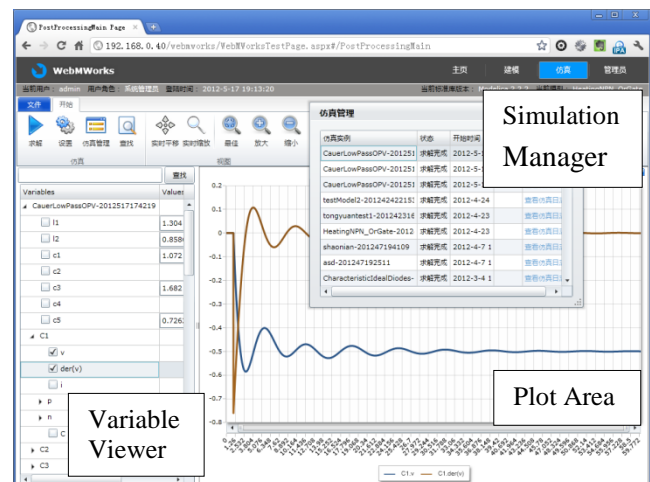


Figure 7: Post-Processing Environment in Browser

The *simulation manager* can monitor and control which state the model is in the compiling and simulation queue. The possible states are: queuing, processing, finishing and failure. Also, the simulation log can be shown in the window.

The variable nodes in variable viewer also adopt the lazy loading mode. When the variable nodes are chosen, the set of dots will be downloaded and shown in the plot Area on the right.

5.3 Modeling Application

In the graphical modeling client, the graph which is shown in icon/diagram view defined by Modelica annotations. To get Modelica annotation, the client of WebMWorks used .NET platform to re-implement some functions of modeling module of MWorks, call the interface of compiling through p/invoke, and at last realize loading Modelica model library. The package of the function which the client modeling need is achieved by WCF, and can be shown to provide visual modeling service.

The modeling application can provide the WCF interface which can get SVG of icon view of model (scalable vector graphics, which is based on XML).

For example, *GetIconSVG (Modelica.Electrical.Analog.Basic.Resistor)* can get the SVG of icon view of the Resistor Model.

```
<?xml version="1.0" encoding="utf-8" standalone="no" ?>
<svg width="200" height="200">
  <g transform="matrix(1 0 0 -1 100 100)">
    <line x1="-60" y1="50" x2="60" y2="50" style="stroke:rgb(0,0,255);fill:none;stroke-width:0.5;" />
    <line x1="-40" y1="30" x2="40" y2="30" style="stroke:rgb(0,0,255);fill:none;stroke-width:0.5;" />
    <line x1="-20" y1="10" x2="20" y2="10" style="stroke:rgb(0,0,255);fill:none;stroke-width:0.5;" />
    <line x1="0" y1="90" x2="0" y2="50" style="stroke:rgb(0,0,255);fill:none;stroke-width:0.5;" />
  <g transform="matrix(1 0 0 -1 0 0)">
    <text x="-78.8363876342773" y="53.113208770752" font-size="48.3018867924528" style="fill:rgb(0,0,255);">%name</text>
  </g>
  <g transform="matrix(7.54979e-009 0.1 0.1 -7.54979e-009 9.53674e-007 100)"
  isConnector="true" name="p" type="Modelica.Electrical.Analog.Interfaces.Pin" description=""
  parent="Modelica.Electrical.Analog.Basic.Ground">
    <rect x="-100" y="-100" width="200" height="200" style="stroke:rgb(0,0,255);fill:rgb(0,0,255);stroke-width:0.5;" />
  </g>
</svg>
```

Figure 8: Icon SVG of the Resistor model

The client can call the interface of sever through the proxy of WCF to get the SVG which then will be displayed in the icon view of *Resistor* model.

5.4 Multi-task implementation

Compilation and calculation of the model based on Modelica are always time-consuming. In the case of multi-user, when a model is compiled by compiler, others cannot be compiled along. (MWCCompiler does not yet support compiling more than one model at the same time).

So, the compilation and calculation of WebMWorks process in queue is formed to solve concurrent calculation and compilation of multi-tasking and multi-user. In this method, user can submit multi-simulation task in the client, and quit the system after submitting tasks, which explore the full benefits of the web systems

6 Conclusions and future work

This article presents a general web-based modeling and simulation environment: WebMWorks. The design and implementation of the environment are described. Through transplanting the stand-alone tools of modeling and simulation for Modelica into the Web, the usage scenarios are greatly expanded:

For the enterprise and research organizations, the co-design and co-simulation could be achieved, based on the tools of modeling and simulation on web and through combining with model sharing and workflow management.

For individuals, the characteristics of the Web system, such as cross-platform capability, wide accessibility, would improve the efficiency of modeling and simulation. Model sharing and reuse, multi-tasking also helps to improve the speed of the modeling and simulation.

But, the prototype of WebMWorks lacks of several functions compared with MWorks. In the future, it should be improved gradually in the following areas.

- Enforce the capability of data transport, especially compression and decompression of the transporting data;
- Add the function of highlighting and code folding for textual modeling;
- Improve safety of the model store;
- Add the function of 3D visualization in post-processing.

References

- [1] Tummescheit H. Design and Implementation of Object-Oriented Model Libraries using Modelica. Lund, Sweden: PhD thesis, Department of Automatic control, Lund Institute of Technology, 2002.
- [2] James Byrne, Cathal Heavey, P.J. Byrne. A review of Web-based simulation and supporting tools. *Simulation Modelling Practice and Theory* 18 (2010) 253–276.
- [3] Zhou Fan-li, Guo Jun-feng, Zhao Jian-jun, Chen Li-ping. Reusability of Modelica Simulation Model. *System Simulation Technology & Application (Vol 11)*.
- [4] Oscar Duarte. UN-VirtualLab : A web simulation environment of OpenModelica models for educational purposes. *Proceedings 8th Modelica Conference, Dresden, Germany, March 20-22, 2011*
- [5] Eva-lena Lengquist S , Susanna Monemar , Peter Fritzson , Peter Bunus DrModelica – A WebBased Teaching Environment for Modelica In *Proceedings of the 44th Scandinavian Conference on Simulation and Modeling (SIMS'2003)*
- [6] Mohsen Torabzadeh-Tari, Zoheb Muhammed Hossain, Peter Fritzson, Thomas Richter. OMWeb – Virtual Web-based Remote Laboratory for Modelica in Engineering Courses. *Proceedings 8th Modelica Conference*

- rence, Dresden, Germany, March 20-22, 2011.
- [7] Zhengyin Shi, Shenglin Zhao, Shan-an Zhu. An Internet-based Electrical Engineering Virtual Lab: Using Modelica for Unified Modeling. Communication Software and Networks (ICCSN), 2011 IEEE 3rd International Conference on 27-29 May 2011.
 - [8] Sven Meyer zu Eissen, Benno Stein Realization of Web-based simulation services Computers in Industry 57 (2006) 261–271
 - [9] Björn Johansson. COMPUTATIONAL METHODS APPLIED TO MODELICA SIMULATION MODELS IN A WEB BASED FRAMEWORK. Proceedings of IDETC/CIE ,September 24-28, 2005, Long Beach, California USA
 - [10] F.-L. Zhou, L.-P. Chen, Y.-Z. Wu, J.-W. Ding, J.-J. Zhao, Y.-Q. Zhang. MWorks: a Modern IDE for Modeling and Simulation of Multi-domain Physical Systems Based on Modelica. *Modelica 2006, Vienna Austria, September 2006*
 - [11] Sven Meyer zu Eissen, Benno Stein. WEB-BASED SIMULATION:APPLICATION SCENARIOS AND REALIZATION ALTERNATIVES. Proceedings of the TMCE 2004, April 13–17, 2004
 - [12] http://en.wikipedia.org/wiki/Model_View_ViewModel.

Using BCVTB for Co-Simulation between Dymola and MATLAB for Multi-Domain Investigations of Production Plants

Irene Hafner¹, Matthias Rössler², Bernhard Heinzl², Andreas Körner¹,
Felix Breitenecker¹, Michael Landsiedl³, Wolfgang Kastner²

¹) Vienna University of Technology, Institute of Analysis and Scientific Computing
Wiedner Hauptstr. 8-10, 1040 Wien

²) Vienna University of Technology, Institute of Computer Aided Automation
Treitlstr. 3, 1040 Wien

³) dwh Simulation Services
Neustiftgasse 57-59, 1070 Wien

Abstract

This paper discusses the cooperative simulation of models implemented in Modelica, Simscape, Simulink and MATLAB for the aim of energy optimization in cutting factories. To simulate the thermal processes in production halls, the machines and the room itself have to be modelled in varying detail. To achieve a quite accurate comprehensive model, the individual machines and the room are modelled in different software and then simulated with the co-simulation tool BCVTB, which stands for *Building Controls Virtual Test Bed*. The communication between the individual models requires a lot of preparative work and as can be seen at the end of the paper, it works fine for a fixed communication time step but is not possible with a continuous synchronization for all given software. Still, the possibilities of co-simulation with BCVTB can be found sufficient for the needs of thermal processes which react very slowly and not in time steps of hugely differing dimensions respectively, but require a period of time which can easily be approximated small enough for a certain scenario.

Keywords: co-simulation; BCVTB; energy optimization; Dymola/Modelica

1 Motivation

Nowadays it has become more and more important to be able to simulate models with partial models of different complexity and differing requirements regarding solver algorithms, step sizes and other model-specific properties. To meet these requirements,

models of such complexity are approached via co-simulation. Co-simulation stands for “Cooperative Simulation”. One can tell from the name that the aim is to simulate separate models and let them communicate and synchronize to certain points in time given by an overall simulation which lets all partial models cooperate.

The aspects discussed in this paper are part of the INFO (Interdisziplinäre Forschung zur Energieoptimierung in Fertigungsbetrieben) project which is promoted by the Austrian Research Promotion Agency (FFG). Its aim is to optimize the energy consumption in cutting factories. Therefore it’s necessary to simulate the thermal processes in production halls. Since all different machines in one production hall require individual modelling approaches, certain solvers and even different software, this problem is approached with co-simulation.

Via the Ptolemy-based co-simulation tool BCVTB (Building Controls Virtual Test Bed), a room model implemented in Modelica, machines implemented in Modelica, Simscape and Simulink as well as a MATLAB data model of the measured heat emission of a machine are co-simulated. Figure 1 gives an overview of the desired communication between the individual simulators.

2 Building Controls Virtual Test Bed

The Building Controls Virtual Test Bed was designed at the University of Berkely to allow the communication of the simulators Ptolemy, EnergyPlus, Dymola, Matlab, Simulink, Radiance and BACnet. BCVTB

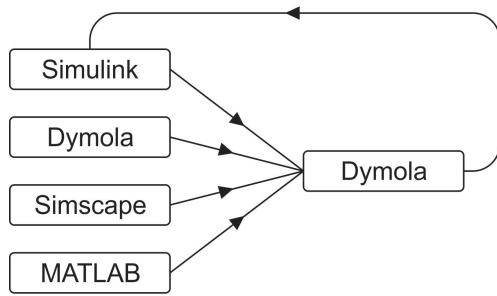


Figure 1: Overview of the Intended Communication between the Individual Simulators

resembles the Ptolemy interface but offers additional blocks (*actors*, as they're referred to in Ptolemy) and on the other hand lacks Ptolemy elements which are not necessary for the use of co-simulation, which BCVTB has been developed for. Though some of the Simulators would be able to interact without the BCVTB interface (like Dymola and Simulink), the use of different step sizes or even solver algorithms is only possible with co-simulation.

To control the synchronization of the individual simulators, BCVTB provides certain so-called *directors*. The *Continuous Time Director* (CT) allows the user to choose a variable step solver (explicit RK23 or RK45) for the total simulation as well as setting solver options like the maximum step size or the error tolerance (see Fig. 2).

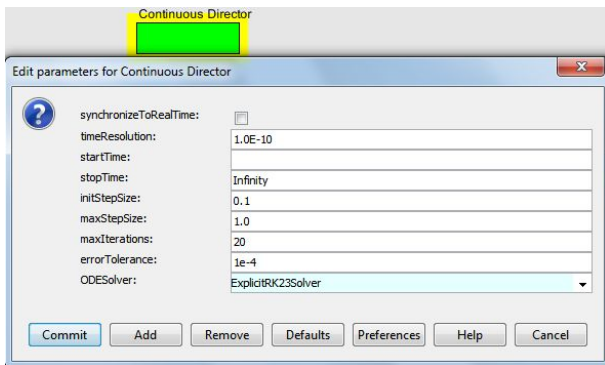


Figure 2: Continuous Time Director

If it's sufficient for a model to synchronize all partial models at predefined fixed time steps, the *Synchronous Data Flow* (SDF) director can be used. All properties of the SDF Director can be seen in Fig. 3:

From the BCVTB interface, the different simulators have to be accessed with *Simulator* actors. These actors establish the communication among the individual Simulators via BSD sockets, which also have been developed at the University of Berkeley and are used for inter-process communication (see [3] for

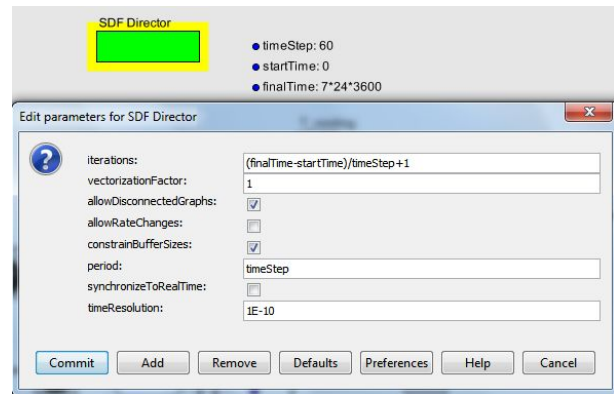


Figure 3: Synchronous Data Flow Director

further information).

All values needed by a simulator have to be connected to the input port, which allows multiple inputs; all values which the simulator returns to BCVTB at each synchronization time step can be accessed from the output port of the simulator actor. The options of the simulator actor (see Fig. 4 for a simulator actor accessing MATLAB) define the simulator to be called as well as options for the simulator, the execution file, the path where it can be found and a parameter *socketTimeout*. This parameter defines how many milliseconds BCVTB has to wait for the simulator to respond before canceling the simulation and returning an error. If a BCVTB model fails due to this *socket time out*-error, there is either an error in the partial model or it simply takes longer than the given *socketTimeout* to load and thus is not able to respond early enough. Hence it is important to choose an adequate amount of time for complex partial models.

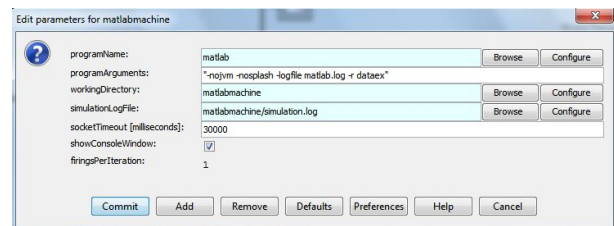


Figure 4: Simulator Actor Accessing MATLAB

2.1 Communication between Dymola and BCVTB

To enable the communication of BCVTB with Dymola, the developers of BCVTB have implemented the Modelica Buildings Library which provides a BCVTB block (see Fig.5).

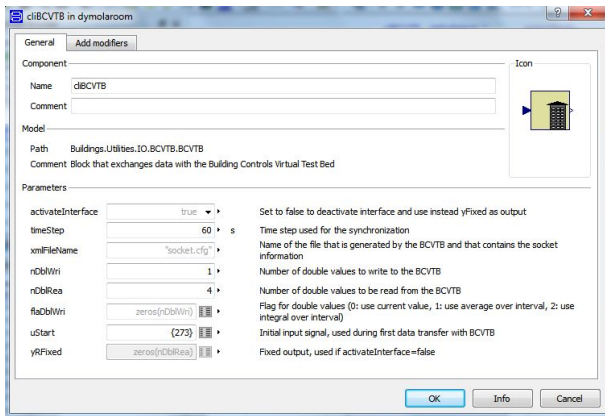


Figure 5: BCVTB Modelica Block enabling the Communication between Dymola and BCVTB

Inputs to the block are all values to be transferred from Dymola to BCVTB, outputs are all values needed from the BCVTB. In the block properties the time steps at which Dymola has to synchronize with BCVTB can be defined by setting the parameter *timeStep* to the desired value.

nDbIWri defines the number of values Dymola returns to BCVTB and *nDbIRea* stands for the number of values Dymola will receive from BCVTB at each synchronization time step. All data received from BCVTB is kept constant between the synchronizations.

The parameter *uStart* stands for the value which is returned to BCVTB at the very first synchronization.

2.2 Communication between MATLAB and BCVTB

In MATLAB, the first step necessary to enable the communication with BCVTB is to create a socket connection via

```
sockfd = establishClientSocket('socket.cfg');
```

Further the following values have to be exchanged with BCVTB at every desired time step by calling

```
[retVal, flaRea, simTimRea, dblValRea] = ...
exchangeDoublesWithSocket(sockfd, flaWri, ...
length(u), simTimWri, dblValWri);
```

retVal, *flaRea*, *simTimRea* and *dblValRea* represent the values obtained from BCVTB which can now be used in the MATLAB function. MATLAB has to submit *sockfd*, *flaWri*, *length(u)*, *simTimWri*

and *dblValWri* to BCVTB. Before completely exiting Matlab, the socket is closed with

```
closeIPC(sockfd);
```

2.3 Communication between Simulink and BCVTB

For the communication with Simulink, BCVTB also offers a preimplemented block. Inputs are again all values from Simulink to be sent to BCVTB and outputs are the values Simulink needs from BCVTB. The underlying subsystems can be seen in Fig. 6.

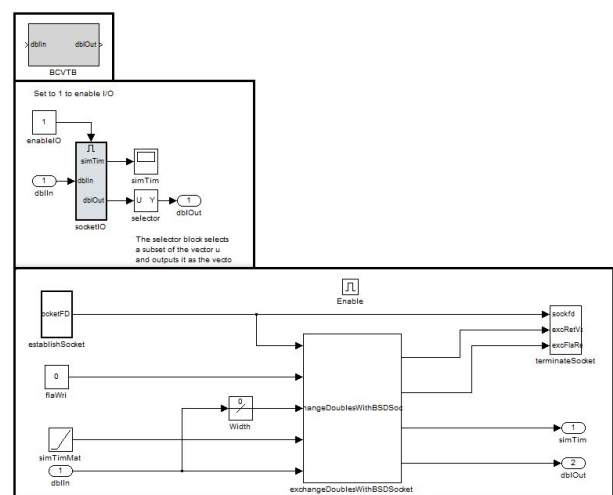


Figure 6: BCVTB Simulink Block enabling the Communication between Simulink and BCVTB

In contrary to the BCVTB block for Dymola, the time step for synchronization cannot simply be defined by a block parameter. For all preimplemented examples BCVTB offers, the time step of the Simulink solver is chosen fixed and equal to the BCVTB time step so there's no problem since the synchronization automatically takes place at the correct time.

To be able to benefit of one of the main advantages of co-simulation - the usage of different solvers and different step times - additional programming work has to be done. To fulfill this purpose, the BCVTB block is put in an *If Action Subsystem* which is activated only if the time step of the BCVTB director is crossed. In case of a SDF director, which means a constant time step, the maximum time step for the solver in Simulink is set to this constant and the time in Simulink modulo the BCVTB time step is compared in every Simulink time step. If the Simulink time crosses the BCVTB time step, the modulo value changes and after zero-crossing detection to evaluate the return value at the

desired time within a certain tolerance, the If Action Subsystem is activated and the exchange takes place (see also section 3.3 and Fig. 11). If a CT director is used in BCVTB, the time step varies and can't be foreseen, so the time in BCVTB is compared to the time in Simulink and at every time step iterated this way the subsystem is activated by sending a discrete impulse at these points in time.

3 Model Description

The model described in this paper uses Dymola/Modelica, MATLAB, Simscape and Simulink apart from the main model in BCVTB. It's purpose is to demonstrate the thermal processes in a production hall. The hall itself is modelled in Modelica. The different machines are implemented in Modelica, Simscape and as simple data model in MATLAB. To obtain a bearable room temperature for human workers which possibly enter the hall, a controller is implemented in Simulink. The waste heat of the machines and the cooling heat from the controller are transferred to the room model at each synchronization time step via the BCVTB interface. The BCVTB model can be seen in Fig. 7:

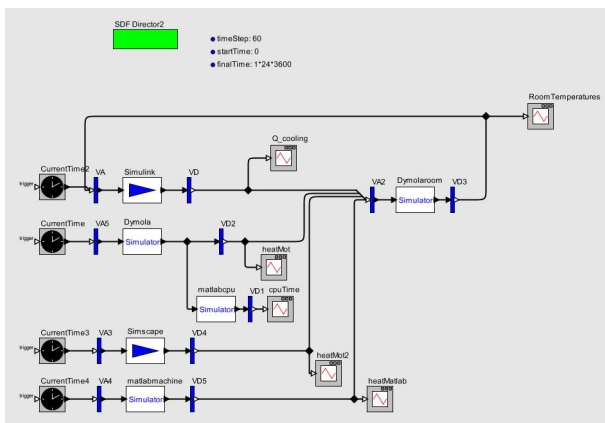


Figure 7: Model for Synchronization in BCVTB

The model is supervised by a SDF Director which demands so-called *firing* of the individual simulators every 60 seconds. The stop time can be defined by the parameter *finalTime* in seconds. Since the machines don't need any values from BCVTB apart from the time, they receive the current simulation time only. The simulator actors *Simscape*, *Dymola* and *Matlab*, which enable the communication with the respective machine models, return the heat outputs which are then sent to the room model called by the *Dymola*-

room simulator actor. The output of the *Dymolaroom* simulator is a temperature measured in one of the compartments of the room (see section 3.1) which is then sent to the controller represented by the *Simulink* simulator actor. The output of the controller is again sent to the room model and treated as a heat source. To obtain a better documentation of the simulation process, the model also communicates with a MATLAB function which stores the elapsed cpu time to an excel-file and additionally sends it to BCVTB for immediate visualization. The cpu time taken by the communication and execution of the m-file realizing the cpu documentation can be regarded negligible in comparison to those of the other partial models, which are way more complex and therefore expensive.

3.1 Room Model in Dymola/Modelica

The model of the production hall is realized as a compartment model. Each thermal compartment basically represents a cuboid with a certain heat capacity and conduction at the surfaces. The graphical model and all parameters of a thermal compartment can be seen in Fig. 8.

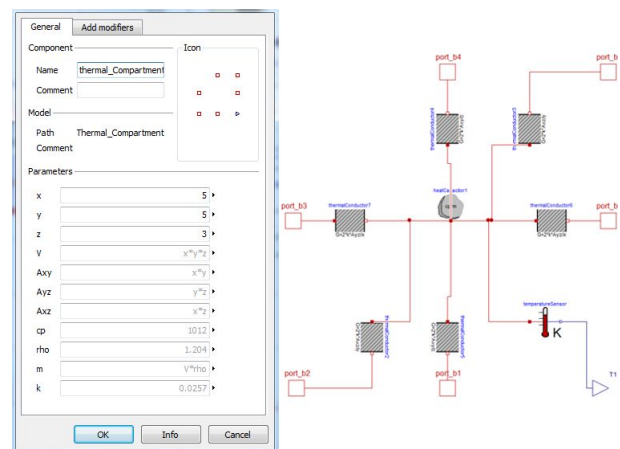


Figure 8: Model for a Thermal Room Compartment Implemented in Modelica - Parameters and Graph

The model of the production hall consists of six thermal compartments at $5 \times 5 \times 3 \text{ m}^3$ each (see Fig. 9).

The heat emitted by the machines and the regulation heat flow from the controller can be accessed at the output port of the BCVTB block and are transferred as prescribed heat flow to the compartments where the machines are found in the production hall. The temperature measured in one of the compartments is returned to the BCVTB model and further to the controller.

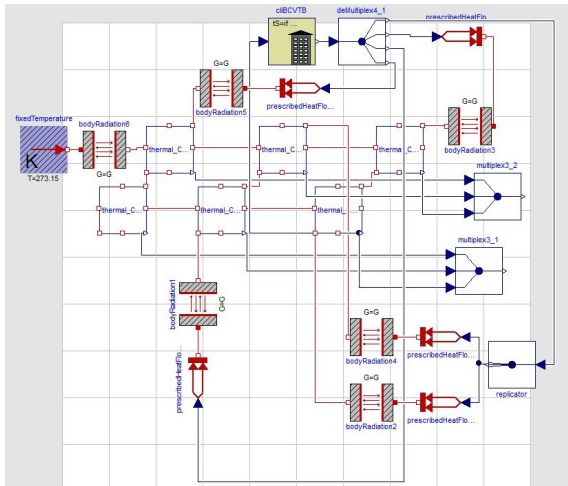


Figure 9: Model Graph for a Machine Hall Implemented in Modelica

3.2 Machine in Dymola/Modelica

Since the main focus lies on coupling the individual models, the machines involved are held rather simple. The machine implemented in Modelica consists basically of a DC motor. Since version 3.2 of the Modelica standard library, the heat dissipated in an electrical circuit can be used in a thermal system by activating an optional heat port at certain components. The electrical energy lost at the resistor of the model is converted into thermal energy, which is measured as heat flow from the resistor heat port to the room represented by a heat capacitor.

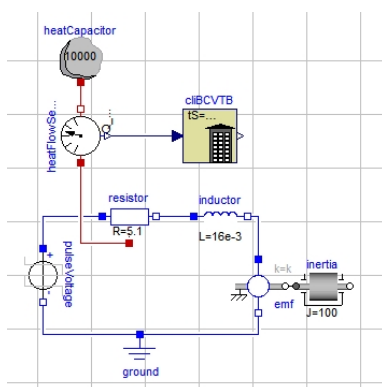


Figure 10: Model Graph of a DC Motor Implemented in Modelica

To simulate different loads by machines which don't run 24 hours a day, the voltage applied to the voltage source is chosen as pulsating with 320V at working hours and 0V at night.

3.3 Machine in Simscape

The machine in Simscape is represented by a motor similar to the one implemented in Modelica. To use the waste heat emitted at the resistor, the rated power is manually calculated from the voltage drop and sent to a thermal system as heat flow. Again, the working hours of the machine are set via the voltage source.

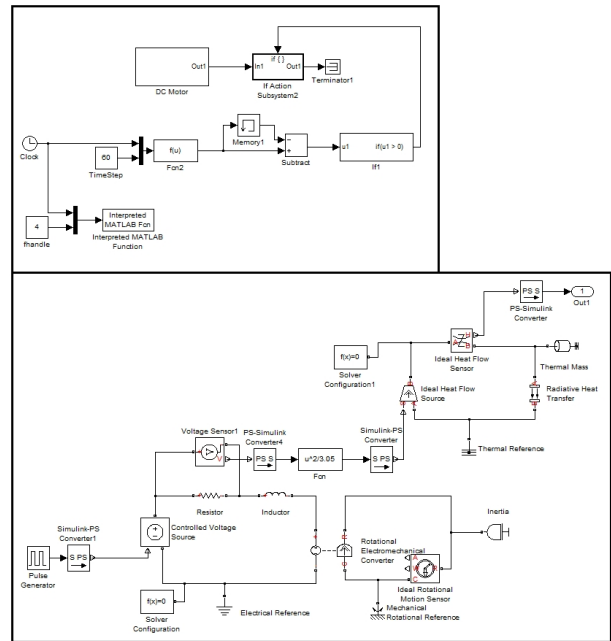


Figure 11: Model of a DC Motor Implemented in Simscape

3.4 Controller in Simulink

The temperature control is realized rather simply. The model gets the temperature measured in one of the *Thermal Compartments* of the Dymola room model and compares it to the desired room temperature. If the room is more than one Kelvin too warm (cold resp.), the control returns minus (plus resp.) 100W heat flow to two room compartments.

3.5 Data Model in MATLAB

The data model in MATLAB is rather simple. The heat emission of a machine over one day is read out of an excel-file and returned to the BCVTB model and further the Dymola room model at each time step.

4 Simulation Results

The model is simulated for one day to show the behaviour of the model for this time span. At 8 a.m.

all machines start working and the room temperature (measured in one compartment for the cooling system) which can be seen in Fig. 12 begins to rise. As soon as the room temperature reaches 294.15 K, the control starts cooling.

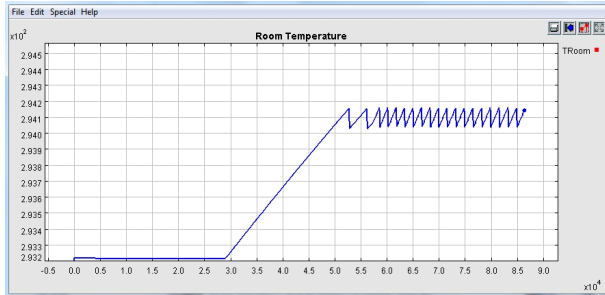


Figure 12: Progress of the Temperature in One Compartment

The temperature graph of all compartments is shown in Fig. 13. The temperature measured in the compartment shown above corresponds to the green one in Fig. 13. One can easily see that the compartments containing machines (blue, red and pink) respond much more quickly than the others.

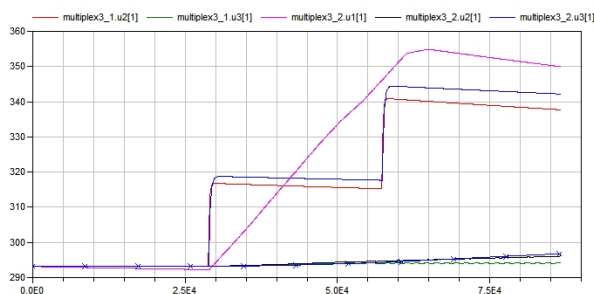


Figure 13: Progress of the Room Temperature in All Compartments

The heat emitted by the individual machines is demonstrated in Fig. 14. Turning down the machines implemented in Simscape and Dymola causes a step response similar to the one caused by switching them on. The measured heat emission transferred to BCVTB with MATLAB shows a rather permanent emission during working hours.

A very important result of the simulation is the documentation of the individual step sizes. Figure 15 shows the solver time steps between two synchronization references of the simulation. For the simulation of the machine and the room model in Dymola, the *Dassl* solver is used. The machine model in Simscape is simulated with *ode15s*, a variable step solver for stiff systems. Since the control in Simulink only deals with

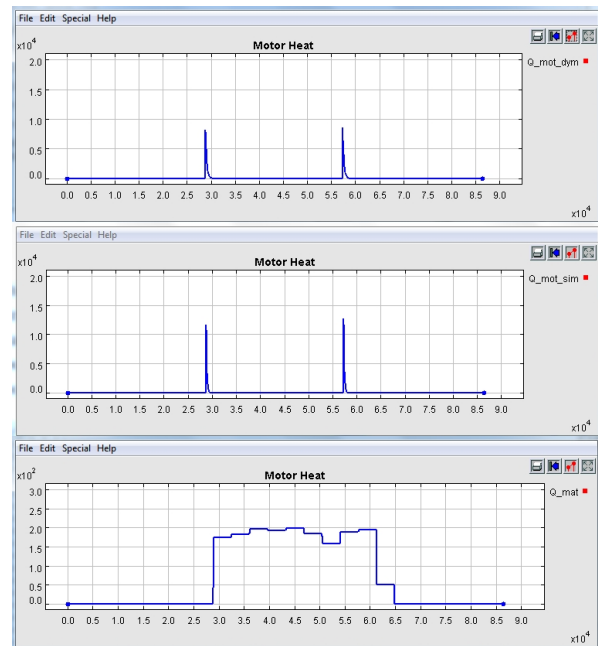


Figure 14: Heat Emitted to the Room by the Machines (Simulated in Matlab, Simscape and Dymola) over One Day

discrete states, *variable step discrete* is chosen for the simulation. The fixed time step for synchronization in the BCVTB model is set to 60 seconds.

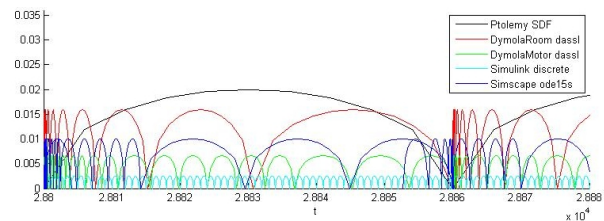


Figure 15: Plot of the Different Solver Time Steps between Two Synchronization References

One of the most important advantages of co-simulation becomes very obvious in this plot: The time steps in the machines, which also differ clearly from each other, are significantly smaller than the time steps of the room in Dymola. This makes perfect sense due to the fact that for the machines systems of equations out of electrical and mechanical circuits have to be solved. Since electrical and mechanical components interact much faster than thermal ones, the underlying systems require accordingly smaller time steps. Fig. 16 shows the different solver steps made in a very small interval around a synchronization reference. This clearly points out the redundant steps made by the Simscape solver to iterate the accurate time to communicate with BCVTB.

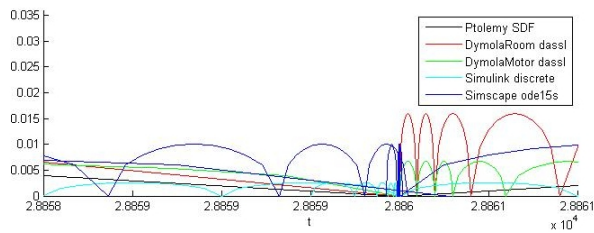


Figure 16: Plot of the Different Time Steps at a Synchronization Reference

Finally, the progression of the room temperature during the simulation of the same model over three days is shown in Fig. 17. Of course the very simple way of cooling can't prevent the temperature from boundless rising in compartments with machines.

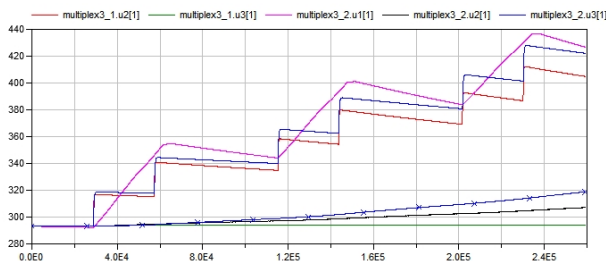


Figure 17: Progress of the Room Temperature in All Compartments over Three Days

5 Conclusion

At the first impression, BCVTB seems like a quite advanced tool to enable cooperative simulation in a rather easy way. It's true that after successfully installing compatible releases of every software required and modifying the given synchronization tools to even allow differing solver time steps, coupling of several partial models in a BCVTB model can be realized without huge modifications.

On the other hand it's not possible to let models communicate with BCVTB at variable time steps with the given BCVTB blocks. In Simulink the communication at time steps which aren't known before can be realized by activating a subsystem containing the BCVTB block. To also achieve this in Dymola, most parts of the given BCVTB block would have to be rewritten. What's more is that between two synchronization time steps all values from BCVTB are extrapolated uniformly so depending on the actual graph and the synchronization step size, the single errors could sum up to an amount which causes the model to fail any validation. For the described use in thermal systems which

react very slowly, co-simulation with BCVTB might be considered sufficiently accurate, but to achieve a valid co-simulation which requires precise or at least reliable approximations with arbitrarily small errors, other possibilities of co-simulation will have to be considered.

6 Outlook

In the course of this project, the limits of co-simulation with BCVTB will be further explored considering the complexity of individual models as well as the amount of partial models involved. Additionally, a room model in the building energy simulation program EnergyPlus will be implemented and further compared to the room model in Dymola to depict the advantages of the different software regarding co-simulation with BCVTB as well as the behaviour as thermal model for a production hall.

Acknowledgement

This work was partially supported by the Klima- und Energiefonds of the Austrian Federal Government within the Neue Energien 2020 program (FFG Project No. 825384).

References

- [1] Wetter M. Building Controls Virtual Test Bed User Manual Version 1.1.0. Berkeley, California: Building Technologies Department, Environmental Energy Technologies Division, Lawrence Berkeley National Laboratory, 2012. Available from: <http://simulationresearch.lbl.gov/bcvtb>
- [2] Heinzl B., Rössler M. et al.. Studies on Multi-Domain Modelling and Thermal Coupling of a Machine Tool. Winterthur, Switzerland: ASIM 21. Symposium Simulationstechnik, 2011 ISBN: 978-3-905745-44-3
- [3] Stevens W.R., Fenner B., Rudoff A.M.. Unix Network Programming: The Sockets Networking API, Vol 1. Addison-Wesley Professional, 2004 ISBN: 9780131411555
- [4] Modelica Buildings Library V1.1. Available From: <http://simulationresearch.lbl.gov/modelica>

FEM models in System Simulations using Model Order Reduction and Functional Mockup Interface

Andreas Gödecke*, Monika Mühlbauer†, Jörg Nieveler, Iason Vittorias‡, Thomas Vontz
Siemens AG, Corporate Technology
Otto-Hahn-Ring 6
81739 Munich, Germany

Abstract

The integration of a three-dimensional FEM model (ANSYS) in a dynamic, component-based system simulation tool (CoSMOS) is described. In order to avoid high simulation times of a direct co-simulation while maintaining the relevant details of the FEM submodel at the same time, model order reduction is applied to the FEM model. The reduced submodel is encapsulated in an FMU and finally imported in a system simulation. An example use case is presented to demonstrate the workflow.

Keywords: FMI, model exchange, model order reduction, CoSMOS, system simulations

1 Introduction

The need to describe complex dynamic systems, which involve several physical disciplines of potentially different timescales and levels of detail, increases continuously - e.g. almost any mechatronic design or power generation process falls into this category. Due to the interdependence of subsystems, a sequential development process is very time and cost consuming and requires many iterations. To make it more efficient, system simulations are used but their realization is hindered by a large variety of tools and models applied for and best suited to the different subsystems. An expensive, specific coupling for every pair of tools can be avoided by a standardized interface, like the Functional Mockup Interface [1], [5]. It has been introduced at the Siemens AG to combine several component-based system submodels, see [10]. In this contribution, we show a workflow that uses FMI for model exchange (version 1.0) to transport relevant

information from very detailed, three-dimensional FEM models. These are often performed in the detailed design phase of subsystems but bear a level of detail too high for system simulations, which is connected to long simulation times and large data amounts. To overcome these issues, we apply model order reduction to an FEM model, encapsulate it in an FMU and import this in a system simulation, see Fig. 1 for illustration. The latter is done exemplarily in a Siemens-internal, flexible simulation platform called CoSMOS.

In Section 2, we present the workflow in detail, starting from an FEM simulation in ANSYS, describing step-by-step the model order reduction and FMI encapsulation and leading to the CoSMOS simulation. Section 3 gives an example of the approach and Section 4 concludes the paper.

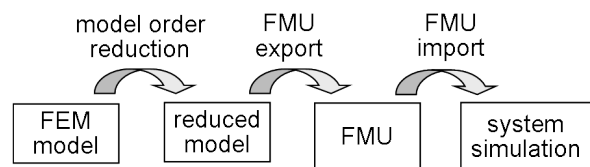


Figure 1: Overview of the workflow proposed.

2 Workflow

2.1 FEM simulation and Model Order Reduction

Model Order Reduction (MOR) techniques can drastically reduce the order and / or dimension of a large dynamical system without considerably sacrificing accuracy. The goal of MOR is to find a projection matrix \mathbf{V} so that the state-space $\mathbf{x} \in \mathfrak{X}^n$ of a full FEM model is projected onto a state-space $\mathbf{x}_r \in \mathfrak{X}^r$, where the dimension r is much smaller, i.e. $r \ll n$, with

$$\mathbf{x} = \mathbf{V} \cdot \mathbf{x}_r + \boldsymbol{\varepsilon} . \quad (1)$$

*andreas.goedecke@siemens.com

†monika.muehlbauer@siemens.com

‡iason.vittorias@siemens.com

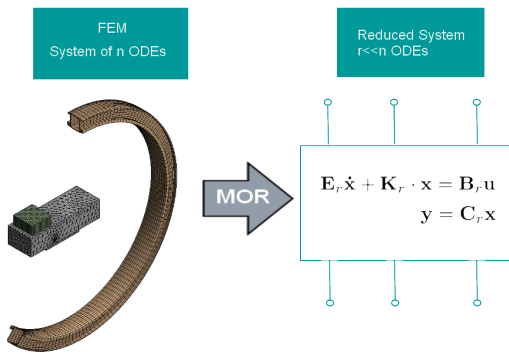


Figure 2: The discretized system, consisting of thousands of ODEs, is projected onto a lower dimension space.

The error difference ε in (1) should be minimal according to a norm specified.

Given the full system matrices from ANSYS and the required new lower dimension, the software MOR for ANSYS [9] generates the reduced order system matrices. Fig. 2 illustrates this workflow. It uses a Krylov-subspace method to find the projection matrix, refer to [8, 6] for more details. For a documentation on how to export the system matrices from ANSYS and how to apply MOR for ANSYS, see [2].

The reduced order matrices can then be imported in Matlab (or other preferred software) where the system can be transformed from the implicit form of Fig. 2 to an explicit one by multiplying with the matrix E_r^{-1} , given that E_r is not singular. The final reduced system then has the following form

$$\begin{aligned} \dot{\mathbf{x}}_r &= \mathbf{A}_r \mathbf{x}_r + \mathbf{B}_r \mathbf{u} \\ \mathbf{y} &= \mathbf{C}_r \mathbf{x}_r . \end{aligned} \quad (2)$$

It is important to note here that the matrices of the state-space (2) are independent of the input \mathbf{u} . This means that the reduced system, once created, can be used for many different inputs in transient or harmonic simulations without requiring any repetition of this process.

The method of order reduction applies to various model domains including electrical, thermal, or even fluid flow domain. Although it can in principle be used for second order systems as well, e.g. most mechanical systems, we focus on first order systems in this first approach, which are additionally linear. Note, that extensions for nonlinear systems exist by either using linearization techniques (or even splitting linear and nonlinear parts), or direct application of nonlinear methodology for model order reduction, e.g. proper orthogonal decomposition [7]. The main disadvantage

of the latter, compared to the linear systems case, is that a full simulation of the system is always required, affecting thereby the required development time. To the best of our knowledge, nonlinear equations are not supported by the software MOR for ANSYS and are not discussed in this paper.

2.2 Exporting a MOR model as FMU

The Functional Mockup Interface (FMI) [1] defines an open interface between numerical simulation tools. A zip-file, called Functional Mockup Unit (FMU) is distributed. It encapsulates a model description and static information in an xml-file as well as the model access in an exchangeable, simulator-independent binary file or source code in C. The present workflow uses FMI for model exchange (version 1.0) [4], which exports the differential equations of the model, relying on the importing simulator to perform the integration.

The FMU Software Development Kit (SDK) by QTronic GmbH was utilized to provide the core FMU functionality. Only a thin wrapper was implemented using Matlab, which performed the task of importing the MOR for ANSYS [9] matrices and converting them into explicit form (2), as discussed in the previous section. Moreover, the wrapper finally creates the C and XML files containing the model equations in the format required by the FMU SDK.

2.3 Import of an FMU in a system simulation tool

CoSMOS (Complex Systems Modeling, Optimization & Simulation) is an in-house simulation platform for dynamic system simulations that has been developed at Corporate Technology, Siemens AG, since 2000, cf. [11]. It is written in C# and C++ and is based on a client-server concept to allow easy tool coupling by using an open and modular architecture. Matlab, Ansys, WinCC, Excel and several optimizers have been considered so far among others.

One default client, called the simulation client, comprises component libraries for one-dimensional simulations in various domains as well as a selection of solvers for systems of differential algebraic equations. Continuous process variables, discrete signals and events can be handled. Results are written to .dcc or SQLite files and can be inspected by a graphical user interface. Dynamic simulations of fresh water and sewage flows, of conveying systems and power plant processes, of electric and traffic networks have

been performed.

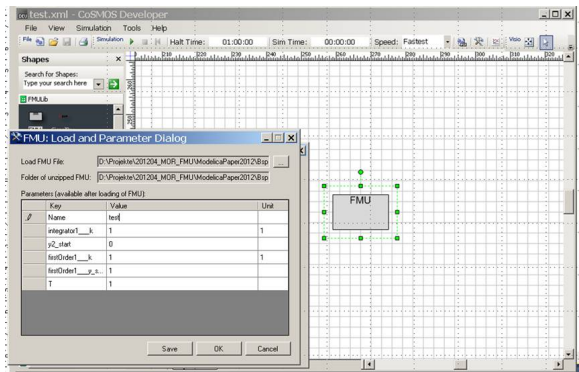


Figure 3: Generic FMU component with parameter dialog upon loading.

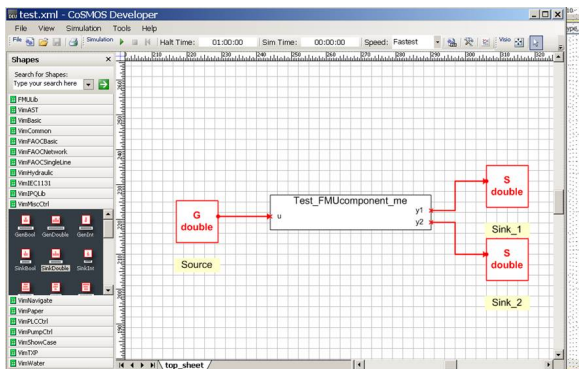


Figure 4: Specified FMU component with known parameters and ports.

Recently, the import of FMUs for model exchange has been implemented. Fig. 3 and Fig. 4 give an overview of the workflow with respect to the graphical user interface. A gray box component that represents a generic FMU is dragged and dropped from the library. After the user has specified the FMU file in the parameter dialog, the box component derives its inputs, outputs and parameters from the FMU xml file containing the model description [4]. This dynamic creation of a component distinguishes an FMU from all other components and its implementation means a considerable effort with respect to the graphical representation. The states of the FMU are added to the degrees of freedom of the overall system, they are updated using the dll-functions provided in the FMU. As FMI version 1.0 does not include Jacobian information yet, the latter is obtained numerically.

3 Example use case: Temperature control of a C-arm device

3.1 Description

In the following, we illustrate our proposed workflow on a thermal model of a C-arm device which is considered in a simple control environment. The C-arm device consists of two main thermal components: An X-ray source, which represents a heat source, and a closed water circuit with the water flowing from the X-ray source to a water tank and forth and back in two channels of the C-arm itself, see Fig. 5. By convection, the heat is transferred to the ambient air. The detailed thermal model of the C-arm device is to be considered in a system simulation which is used to test a very simple two-level control model. The latter observes the material temperature at three positions, where temperature sensors are to be positioned in the real device. If required, an additional fan can be turned on which leads to a decrease of the ambient temperature around the C-arm. The example is kept very simple for demonstration purposes.

3.2 FEM Model

The CAD model of the C-arm is imported in ANSYS (*DesignModeler*), see Fig. 5. The heat source and the tank body are thermally coupled to the C-arm at two areas, the water inlet and outlet. The power profile of

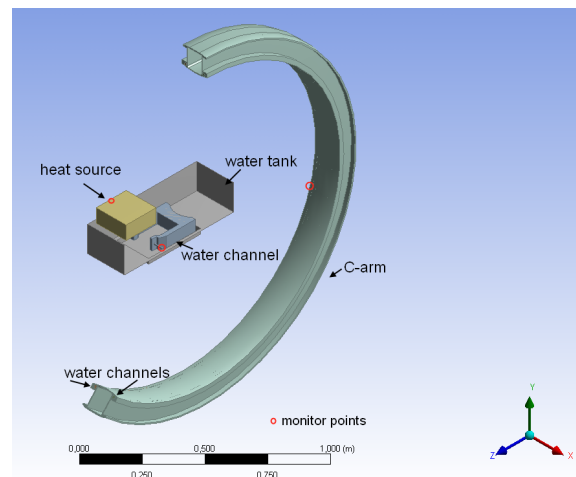


Figure 5: The design of the C-arm device. The two faces of the water tank are hidden to enable the visibility inside it. The red circles indicate the points where the material temperature is observed.

the X-ray source is the input to our model and deter-

mines the amount of heat generated in the heat source. The heat is transferred by conduction through the aluminum wall to the rest of the heat source body and then mainly by convection to the water circuit which has a fixed flow rate of 0.3 kg/s . The colder walls in water tank and C-arm take the heat from the water and discharge it finally to the ambient air by a second convection-dominated process. The convection coefficients are parameters defined as $14 \frac{\text{W}}{\text{m}^2} \text{ } ^\circ\text{C}$ for the C-arm and $0.001 \frac{\text{W}}{\text{m}^2} \text{ } ^\circ\text{C}$ for the water tank. Heat can hence mainly be disposed of in the C-arm itself.

The given geometry and physics data are imported in ANSYS Multiphysics. Accordingly, the load (heat generation) and constraint data (convection coefficients, flow properties) are provided either by using the graphical user interface or by the scripting language APDL. To properly represent the heat transfer caused by the water flow in the ANSYS Multiphysics model, *FLUID116* and *SURF152* elements are used [3].

The full system matrices are extracted using available ANSYS functionality.

3.3 Validation of the reduced model

Model order reduction is applied to the full system matrices using MOR for ANSYS reducing thereby the order of the model from $n = 59990$ to $r = 30$, i.e. $r \ll n$. For validation a comparison of the results was done between a simulation of the order reduced model in Matlab (set up directly and not integrated via an FMU) and an FEM simulation in ANSYS, which is considered to be our baseline. Matlab was chosen, because it was also used to transform the reduced model from implicit to explicit form and to finally build the FMU wrapper.

Three different material temperatures at potential sensor positions of the real device are selected for observation and are indicated by the red circles in Fig. 5: One is located on the heat source, one on the C-arm and one at the inlet of the water tank. The selection of the output nodes affects the structure of the C_r matrix in (2), and naturally, it adjusts the dimensionality of the output vector \mathbf{y} to 3 in this case.

Fig. 6 illustrates the comparison between the FEM simulation and the direct simulation of the reduced model. A huge saving in simulation time is achieved: While the FEM simulation required approximately 6 hours, the reduced system could be simulated in less than a minute. In terms of accuracy, the root mean square error (RMSE) remained very low for all three temperatures with values of 0.19°C , 0.19°C and 0.20°C , respectively. The exchange of the FEM model

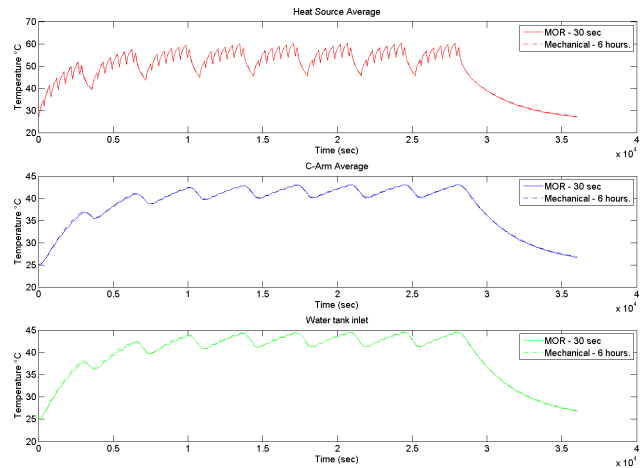


Figure 6: A comparison of a FEM transient simulation in ANSYS with the transient simulation of the reduced order system in Matlab. The power profile is given. Small errors are observed, whereas the reduction in simulation time is drastic, from 6 hours to less than a minute.

with the reduced model is hence justified in a system simulation which considers these three temperatures.

3.4 FMU and overall system behavior

As described in section 2.2, the reduced, thermal model is translated to an FMU which is imported in CoSMOS. The FMU has three inputs: $u1$ represents the power input to the X-ray source, $u2$ and $u3$ the ambient temperatures around the C-arm and the unit of water tank and heat source respectively. The three outputs of the FMU describe the material temperatures at the three intended sensor positions, already discussed in the previous section: $y1$ is located at the X-ray source, $y2$ at the C-arm and $y3$ at the inlet of the water tank.

A very simple two-level controller on the temperature of the heat source is built using the CoSMOS simulation client and its control library, cf. Fig. 7. Its function is tested in the following for a power input of $u1 = 600 \text{ W}$ to the X-ray source. If the temperature measured at the heat source (output $y1$ of the FMU) exceeds 50°C , an additional fan is switched on to full power and reduces the ambient temperature around the C-arm to 0°C (input $u2$ of the FMU). It is assumed that the ambient temperature around the unit of water tank and heat source is not affected (input $u3$). If the material temperature at the heat source falls below 40°C , the fan is turned off.

A modified CHORAL solver (default in CoSMOS) is used to simulate the process where the FMU contributes 90 states. Fig. 8 reveals the behavior of the outputs of the FMU upon simulation. It can be clearly seen that the fan is regularly switched on and off as expected. If the controls are disabled, the temperature at the heat source exceeds 70°C .

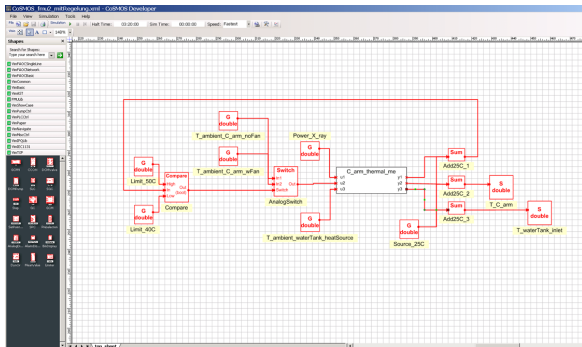


Figure 7: System model in CoSMOS. The thermal model is hidden in the FMU. The controls are built with the CoSMOS control library.

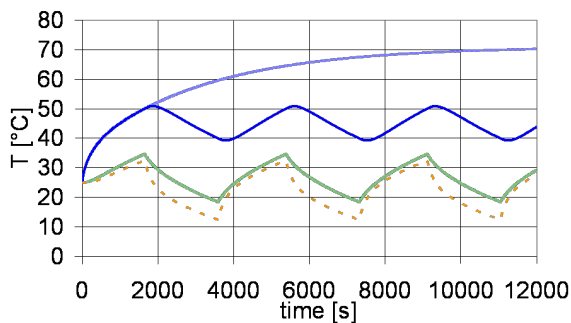


Figure 8: Behavior of the FMU outputs from bottom to top: Temperature at inlet of water tank, at C-arm and at heat source w/ controls and w/ disabled controls

4 Conclusion

In this contribution, we presented and validated a workflow to integrate highly accurate FEM simulations in dynamic system simulations while avoiding long simulation times common to FEM simulations. Model order reduction techniques enabled the creation of state-space representations of the FEM model, and the Functional Mockup Interface was used to transfer the reduced model to the system simulation that was performed with the Siemens internal tool CoSMOS. As an example, the integration of a detailed thermal

model of a C-arm device has been considered in a system simulation which tested a simple control model. The workflow presented applies to any scenario where the accuracy of a lumped model is insufficient for a system simulation, or where overall simulation time is too limited for a direct co-simulation between FEM and system model.

Future work will account for a comparison between various model order reduction algorithms as well as a consideration of stability criteria. The workflow presented up to the FMU generation will be automatized further and the limits of the overall workflow in terms of model complexity and size will be explored.

Acknowledgements

The authors thank Stefan Boschert for providing data to the use case of the C-arm medical device.

References

- [1] <http://www.functional-mockup-interface.org>.
- [2] <http://modelreduction.com/mor4ansys/#Documentat>.
- [3] http://www.cadfem.de/fileadmin/cfappdb/files/Consulting_Flyer_CADFEM_Thermalanalyse_IGBT_Module.pdf.
- [4] http://www.modelisar.com/specifications/FMI_for_ModelExchange_v1.0.pdf.
- [5] T. Blochwitz and M. Otter. The functional mockup interface for tool independent exchange of simulation models. In *Proceedings of 8th Modelica Conference*, Dresden, Germany, 2011.
- [6] R. W. Freund. Krylov-subspace methods for reduced order modeling in circuit simulation. *Journal of Computational and Applied Mathematics*, 123:395–421, 2000.
- [7] P. Holmes and G. Lumley, J. L. and Berkooz. Turbulence, coherent structures, dynamical systems, and symmetry. *Cambridge monographs on mechanics*, Cambridge, New York: Cambridge University Press, 1996.
- [8] E. B. Rudnyi and J. G. Korvink. Review: Automatic model reduction for transient simulation of mems-based devices. *Sensors Update*, 11(1):3–33, 2002.

- [9] E. B. Rudnyi and J. G. Korvink. Model order reduction for large scale engineering models developed in ansys. In *Lecture Notes in Computer Science* v.3732, pages 349–356. Springer, 2006.
- [10] Y. Sun, S. Vogel, and H. Steuer. Combining advantages of specialized simulation tools and modelica models using functional mockup interface (fmi). In *Proceedings of 8th Modelica Conference*, Dresden, Germany, 2011.
- [11] K. Wöllhaf and R. Rosen. A component-oriented simulation approach for industrial plant models: The plantsim simulation tool. *Siemens AG, Corporate Technology, Internal Report*, 2001.

Using Modelica models for Driver-in-the-loop simulators

Mike Dempsey Garron Fish Alessandro Picarelli
Claytex Services Limited

Edmund House, Rugby Road, Leamington Spa, UK

mike.dempsey@claytex.com garron.fish@claytex.com alessandro.picarelli@claytex.com

Abstract

Driver-in-the-loop simulators are increasingly used in Motorsport and Automotive companies to enable engineers and drivers to experience a new vehicle design in a realistic environment before it is built. The use of simulators enables drivers to test a new vehicle and/or control system without having to build a prototype and to carry out those tests in complete safety and in repeatable conditions.

Using Modelica as the development language for the vehicle model within these systems enables rapid model development and the fast evaluation of vehicle concepts. This enables more vehicle concepts to be tested before committing to a prototype build. The use of physical models also ensures that geometry changes and other physical modifications to the concept can be evaluated on the simulator at an early stage.

Keywords: driving simulator, vehicle dynamics, real-time simulation

1 Introduction

Driver-in-the-loop simulators come in a wide range of different formats ranging from the basic workstation simulator through to the high end simulators with multiple projectors and a motion platform with up to 6 degrees of freedom. The aim is to provide a driver with a realistic environment and accurate vehicle response to enable them and the engineers they are working with to evaluate the behaviour and performance of the vehicle design.

Driving simulators are used for a variety of reasons in automotive and motorsport companies. These include driver training, vehicle attribute perception, new designs and the evaluation of new technologies that may affect the concentration/driving pleasure of the driver. Recently there has been an increasing interest in using simulators earlier in the design process to assess the behaviour of new vehicle designs and technologies and to understand how well these systems work together in a vehicle. To meet this re-

quirement it is necessary to improve the model development process so that new vehicle concepts can be quickly modelled using physical models. Dymola and Modelica are ideal for this application and have been used in this way by a number of Automotive and Motorsport users.

2 The Vehicle Model

2.1 Overview

The vehicle model used in a driving simulator has to represent the complete vehicle, accurately predict its behaviour and run in real-time. There is often also a desire to be able to use the same vehicle model in other parts of the engineering process outside of the simulator. Using Modelica to define the vehicle model and Dymola to compile this enables us to meet these requirements as this paper explains.

The vehicle model itself needs to be able to accurately predict the transient performance of the real car. To achieve this it needs to include models of the tyres, suspension, powertrain including both the physical and control aspects of these systems.

A number of commercial libraries have been used in the development of vehicle models for use in driving simulators. These are the Engines and VDLMotorsports Libraries and the following sections describe the use of these libraries to develop a model of a 1.8 litre, 4 cylinder turbo-charged gasoline direct injection engine fitted with a manual gearbox and double wishbone suspension.

This paper will detail 3 of the key aspects to the vehicle model that have been developed to meet these needs: the chassis model; the engine model; and the tyre-road contact model.

2.2 Chassis model

The VDLMotorsports Library [14] is an extension to the Vehicle Dynamics Library [15] developed by Modelon. This library was originally developed for modelling open-wheel race cars where the double

wishbone suspension setup is commonly used together with either pushrod or pullrod actuation of the inboard springs and dampers. The main objective of the library was to provide suspension models for this application that could run in real-time without additional detailed work from the end-users. As part of the intended use of these models was for driving simulators and other trackside tools it was necessary to make sure that the geometry of the compiled models, including all of the physical adjustments normally possible with these suspensions, could be applied through parameter changes without requiring the model to be recompiled. An annotated view of a pushrod suspension showing all the physical adjustments required within the suspension is shown in Figure 1. The shims identified can be defined as a thickness and make a change to the suspension geometry.

To achieve this we developed new implementations of the double wishbone suspension models that shared no common components, but a common architecture with those in the Vehicle Dynamics Library. The major problem to be overcome in defining these suspension models was the non-linear systems of equations that are normally formed when the suspension is created using the Modelica MultiBody library and individual joints (revolute, universal and spherical).

New combinations of aggregated joints [1] were developed to provide an analytic solution to the suspension degrees of freedom which has enabled this mechanism to be implemented without any of the usual non-linear systems of equations. These new joint combinations have been used to define the outboard suspension mechanism as shown in Figure 2 (the animation of this mechanism is shown in Figure 1). The pushrod is defined with an aggregated joint of the form Revolute-Prismatic-Spherical-Prismatic-Universal and is based on the JointUSR in the Modelica Standard Library. The upper wishbone and steering link are a more complex joint structure shown in Figure 3.

In both Figure 2 and Figure 3 the cyan bars on various components represent prismatic adjustments that can be applied to the mechanism to adjust the overall suspension geometry.

In the original baseline model that was created using the Modelica MultiBody library each instance of the suspension model contained 2 non-linear systems of equations of sizes 73 and 57, before symbolic manipulation. In the new suspension models both these non-linear systems of equations are eliminated.

The VDLMotorsports Library also supports the double wishbone suspension setup more commonly found in road cars. These models make use of the

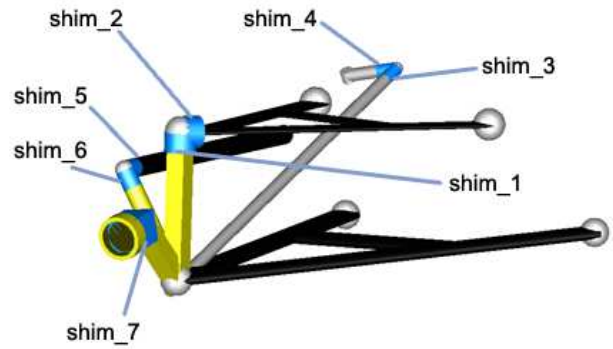


Figure 1: Annotated view of a pushrod suspension highlighting the physical adjustments in the suspension

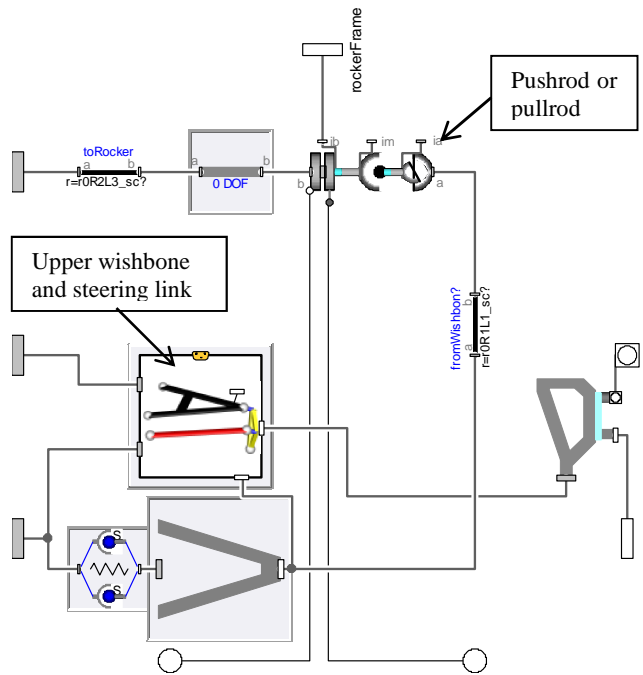


Figure 2: Double wishbone suspension with pushrod/pullrod

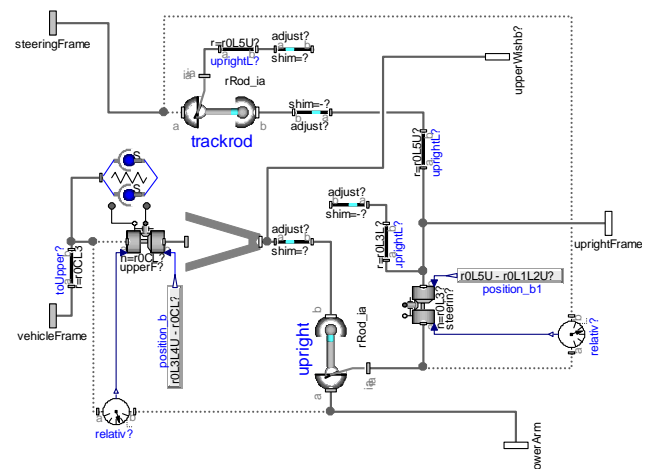


Figure 3: Detailed view of the aggregated joint mechanism defining the upper wishbone and steering link degrees of freedom

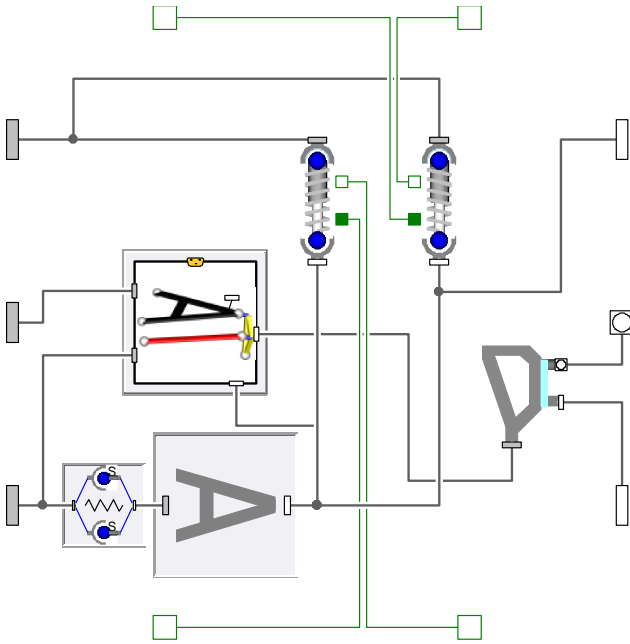


Figure 4: Double wishbone suspension with outboard springs and dampers

same efficient implementation of the suspension but now connect the spring and damper units to the lower wishbone (as shown in Figure 4) or other suspensions links as required.

For this example the vehicle model has been created using the double wishbone suspension with outboard springs and dampers connected to the lower wishbone. This suspension model fits within the Vehicle Dynamics Library suspension architecture and enables the standard steering and anti-roll bar models to be used when applicable. The suspension models are fully adjustable which would enable geometry changes as well as spring and damper rate changes to be assessed when using this model on the simulator.

2.3 Tyre and road contact

The tyre models typically used in driving simulators are based on the Pacejka tyre slip model and as such are essentially single point of contact handling models. The vertical dynamics models are usually implemented as some form of nonlinear spring to more accurately capture the vertical dynamics of the tyre and its input to the suspension. Whilst these single point of contact tyre models give a good prediction of vehicle handling on smooth surfaces they are not really adequate for use on rough surfaces such as those used in driving simulators.

In many cases the simulator road data is based on high-fidelity LiDAR scans of a real road or track surface that captures all the bumps and surface details. This level of detail in the track surface is not

compatible with a single point of contact tyre model because it leads to a lot of noise feeding into the suspension and steering.

In order to make good use of the detailed road surface data a filtering method is introduced into the contact point calculation so that the rough surface is reduced to a single effective contact point. One such method looks at a number of potential contact points underneath the tyre and calculates a resultant contact point and surface normal at which all of the tyre forces are applied.

The screenshot in Figure 5 is taken from a single tyre test rig that is using 5 potential contact points (shown in red) and filtering these to determine one effective contact point (shown in yellow). The filtering is achieved by considering the amount of tyre compression at each potential contact point and then adjusting the position and surface normal of the resultant contact point accordingly. When used with triangular meshed road data some further filtering is then required to avoid sudden jumps in the surface normal as the edges of the different triangles are crossed.

The introduction of these filtering methods in the tyre model is done by replacing the standard Vehicle Dynamics Library contact block with a customized contact model. The development of these filtering methods has driven several enhancements in the Vehicle Dynamics Library ground models which have been implemented by Modelon. These new options now enable the standard implicit contact model, which generates nonlinear systems of equations, to be replaced with an explicit contact model which does not include nonlinear systems of equations.

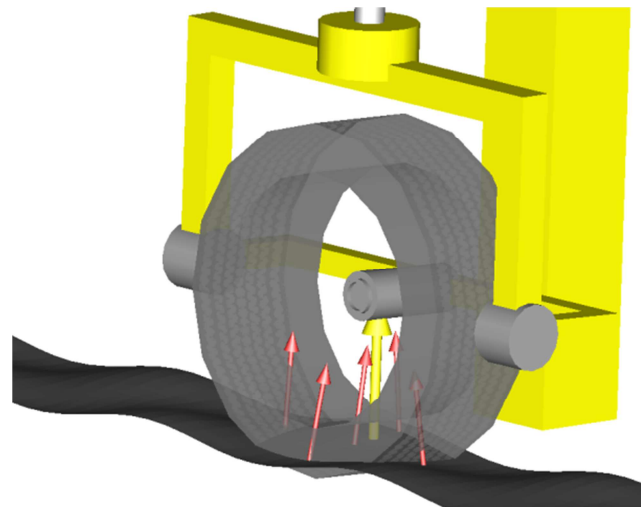


Figure 5: Visualisation of contact point filtering with the potential contact points in red and the resultant contact point in yellow

2.4 Engine model

The engine model used in the test vehicle was developed using the Engines Library [2]. The Engines Library comes in two versions with different capabilities: Mean Value Engine Models (MVEM) which means cycle averaged torque and emissions; and Crank Angle Resolved Engine Models (CAREM) which means crank angle resolution of torque, heat release and emissions. Both versions model the intake and exhaust manifold fluid dynamics and heat transfer with varying levels of detail. The heat transfer models can range from models with no thermal resistance to ones which take into account the flow regime within the particular component [5]. The fluid flow models can range from simple volume models to ones which include the fluid momentum dynamics to analyse fluid pressure pulsations propagating throughout the system and study their effects on the system performance.

For the purposes of CPU time reduction, a surrogate mode is available for both MVEM and CAREM [2],[3] models. The surrogate mode allows the engine model to be reduced in complexity whilst maintaining a high level of accuracy when compared to the non-surrogate mode. Figure 6 shows the comparison of a MVEM model running in both modes and shows that there is little deviation between the results.

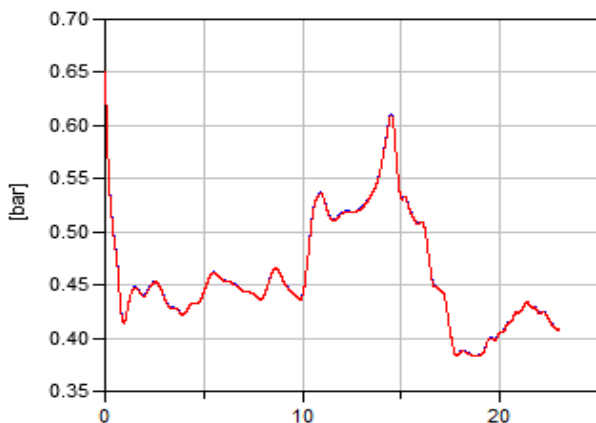


Figure 6: Plot of plenum pressure for a naturally aspirated I4 SI engine with surrogate mode enabled (blue) and disabled (red). In this case the two lines lie almost exactly on top of one another.

All the fluid components in the Engines library are based on the Modelica Fluid library [4] which ensures compatibility with the latter and all derived libraries. The medium model is based on the Modelica Media library but it has been necessary to provide some additional customisations to achieve the level of performance required in this library. The medium model tracks 7 species throughout the air

path of the engine so that fuel mass and the emissions composition can be traced through the engine.

The engine used in this example is a mean value 4 cylinder 1.8l turbocharged spark-ignited gasoline engine with direct injection and producing a peak power of 160kw and peak torque of 225Nm. The transients of the turbocharger, fluids and heat transfer in the air-path and torque output are captured in the results as well as the multi-body behaviour of the system. This model is shown in Figure 7.

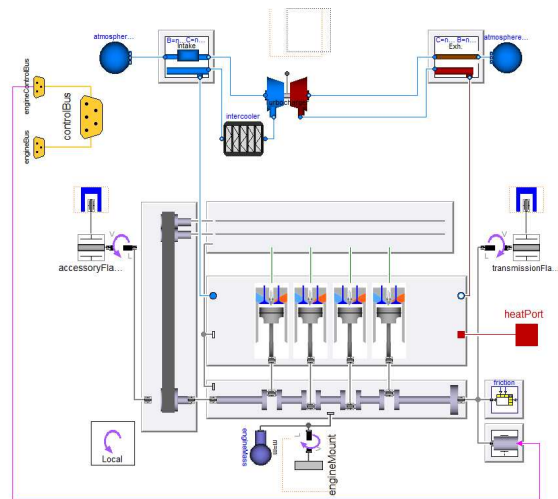


Figure 7: I4 Turbo Mean Value Engine Model diagram layer showing the engine sub-systems. Note that fluid and mechanical connector arrays are used between the components to simplify the diagram layer complexity

In the above model the intake manifold comprises an air flow mass sensor, a turbocharger compressor including ducting, an air to air intercooler, a throttle body, a plenum volume and cylinder head port volumes. The exhaust manifold comprises of the cylinder head ports and primary exhaust system coupled to the turbocharger turbine. Emissions after-treatment systems have been omitted from this engine model as they were not required for the purposes of this experiment; however, their associated pressure drops have been taken into account within the exhaust system. Both intake and exhaust volume models include heat transfer effects from the fluid to the pipe walls.

The Engines library contains two main types of turbocharger turbines and compressors models [6][7][8][10]. There are purely map based models where the mass flow rate and efficiency are functions of the speed and pressure ratio. These models rely on having very good map data available throughout the operating range of the turbine and compressor. The second type of model are based on ellipse curves (e.g. Stodola's law for turbines). In these models ellipse curves are fitted to the available map data to

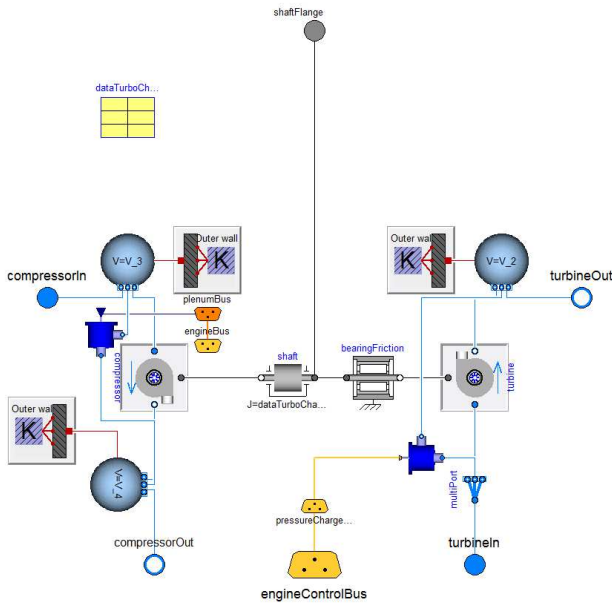


Figure 8: Turbocharger diagram layer showing the layout of the system

describe the mass flow rate through the component. The advantage of these ellipse models is that the characteristics are guaranteed to be smooth.

The engine model described in this paper utilises a turbocharger turbine model based on Stodola's law which calculates the mass-flow rate as a function of the pressure ratio and essentially follows the ellipse law:

$$m_flow_{corrected} = K \cdot \sqrt{1 - \left(\frac{1}{pressureRatio}\right)^2}$$

where K is a constant that scales the turbine characteristics.

The turbocharger compressor uses a similar ellipse law that relates the mass flow rate and pressure ratio through the ellipse equation:

$$1 = \left(\frac{pressureRatio}{a}\right)^z + \left(\frac{m_flow_{corrected}}{b}\right)^z$$

The coefficients a, b and z are defined in tables that are dependent on rotational speed.

The complete turbocharger model is shown in Figure 8. The compressor side (left hand side of the figure) includes intake and outlet ducting with associated replaceable heat transfer models and a pneumatically controlled recirculating pressure relief valve. The turbine side (right hand side) includes outlet ducting and an electronically controlled wastegate to limit the turbine performance. The shaft including the impellers is modelled as a 1D rotational system with bearing friction model.

The ellipse based compressor and turbine improve the model robustness and can rely on less finely resolved compressor and turbine maps than the purely table based variants to achieve similar functionality. Compressor surge is taken care of by supplying a surge line to the model. Special considerations for impeller torque are implemented for operation beyond the compressor surge line.

The basic turbocharger model shown in Figure 8 has been used for both low pressure and high pressure turbocharging applications

2.5 Results

The whole vehicle model including the powertrain was compiled in Dymola using a fixed step solver with inline integration. The inline integration method chosen was a mixed implicit/explicit Euler with step size of 1.25ms. The mixed method was chosen as it gave the best results in terms of expected system behaviour, model robustness and simulation performance. The step size was determined by considering the requirements of the model, the pc processor speed and the frequency that the rest of the simulator system needs to operate at.

In this experiment, the model is compiled with a driver and simulated in Dymola to verify the behaviour. The driver model is open loop for both longitudinal and lateral control and the test consists of an acceleration from rest, shifting through the gears to test the extreme situation for both chassis, wheels and powertrain:

Test sequence:

1. t=0, vehicle standing at rest with engine idling
2. t=2, launch starts, i.e. clutch engagement starts
3. The vehicle accelerates at full throttle through the gears up to a speed of 250km/h

The results of this acceleration test are shown in Figure 9. The main user input required to get such complex models to function in real time is good initialisation. This is assuming that all the work at the component level has already been done to eliminate numerical jacobians and minimise the number of non-linear systems of equations.

The critical areas in these vehicle models are the tyres, suspension and engine initial conditions. Within the VDLMotorsports Library a number of experiments and functions exist that enable the start conditions for the tyres and suspensions so be determined and automatically extracted for use in other simulations. For the engine model, the initial pressures and temperatures within the intake and exhaust system were determined by running the engine at a

constant speed and load operating point that is then used as the start point for the real-time model. The final temperatures and pressures from this steady state experiment are used to define the initial conditions in the real-time experiment.

The results in Figure 9 show the vehicle longitudinal dynamics and the dynamics of the engine components when running the model in real time. We have focused on the intake pressure and the turbo-charger and wastegate [9] performance with the latter used to limit the intake manifold pressure. With such models running in real time, swapping turbo-charger models or tweaking the turbocharger characteristics, enable the user to test different boosting configurations for driveability and performance within a vehicle simulator.

Such models as the one used in this paper provide invaluable and easily accessible detailed information for the engineers in the development phase of the vehicle when there is need to test a variety of options in a short space of time. The transient detail exhibited in the results for the torque generation are of suf-

ficient detail to provide the effects necessary for the evaluation of different engine and powertrain solutions in vehicle simulators.

3 Integrating the physics model and simulator system

3.1 System Overview

A typical driving simulator will consist of several computers each with responsibility for a different aspect of the system. Figure 10 shows the basic architecture of a typical system.

The motion platform usually has a dedicated computer to decide how it should move and receives inputs from the physics model such as vehicle orientation and accelerations.

There is usually a PC (or real-time computer) that is dedicated to the physics model. The physics model receives inputs from both the motion platform via the motion controller and the vision system. The data coming from the motion platform are the driver inputs such as steering, pedal positions, gear, etc. The data coming from the vision system usually includes the environmental conditions and road surface information.

It is also possible to incorporate the real control systems with the physics model. The exact approach varies depending on the objective of the simulator and the hardware available. In some instances a model of the control system is compiled to run on a

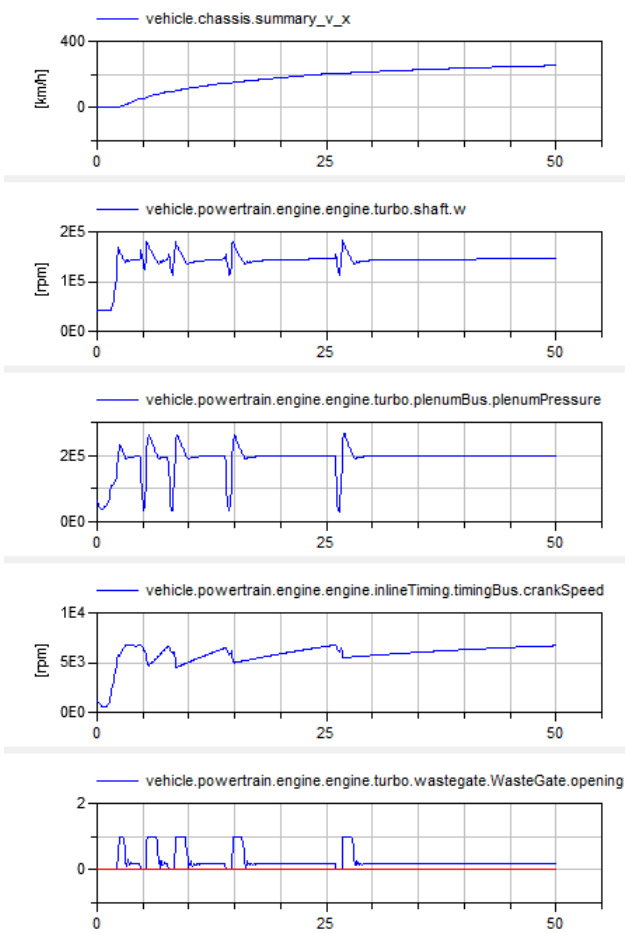


Figure 9: System simulation engine behaviour during the full throttle acceleration test. From top to bottom: Vehicle speed (km/h), turbocharger shaft speed (rpm), absolute plenum pressure (Pa), crank speed (rpm), wastegate (blue) and dump valve (red) opening

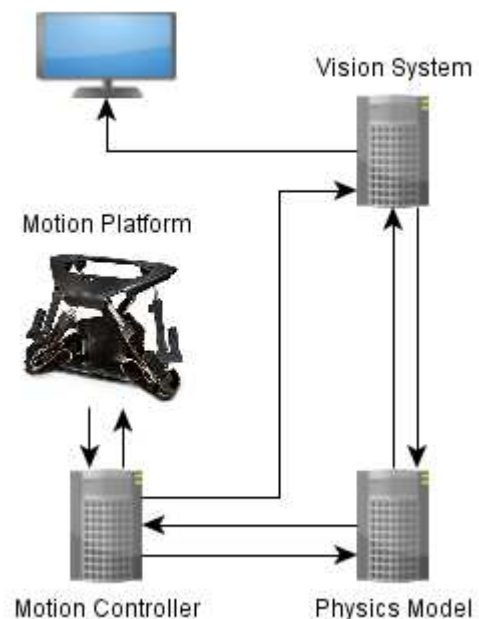


Figure 10: Basic configuration of a typical driving simulator including a motion platform

real-time computer alongside the physics models and in other cases a Hardware-in-the-loop approach is adopted with the real controller connected in to the system.

The vision system itself can consist of multiple computers depending on the actual system configuration. Typically there will be one master computer and then a number of slave machines with at least one computer per projector used.

For a desktop simulator the motion platform and controller is replaced with a steering wheel and pedals that could be as basic as a gaming system or a more specialised system such as steering wheels with high accuracy motors for steering torque assessment.

3.2 Test System Specification

For the example in this paper we have integrated the model described in section 2 within a desktop driving simulator system. We are using rFactor Pro [13] for the vision system and the physics model is running in the McLaren Electronics vTag 310 tool [11]. A Logitech G27 [12] steering wheel and pedals is used for the driving controls together with a single monitor. This configuration makes the driving simulator compact enough to use in the office environment whilst providing a useful tool for evaluating baseline capability of a vehicle or detailed assessment of a control system.

By running the model in the vTag environment we are able to make use of the telemetry system built in to this tool. This means that we can expose the model variables to the telemetry stream and view them in real-time using Atlas [16] (also from McLaren Electronics). This enables logging of the model behaviour for offline analysis.

The use of the vTag environment also enables the control system model to be run alongside the physics model. In Formula 1, for example, it is mandated by the governing body that the complete vehicle control system be developed using the McLaren Electronics tool chain which means it can easily be compiled and run in the vTag environment.

rFactor Pro has an extensive library of scenarios that can be used to test the vehicle. These include LiDAR based race tracks for most of the Formula 1 circuits, North American Indy & NASCAR circuits as well as La Sarthe, Nordschleife and a virtual proving ground with lane-changes, split Mu and low-Mu surfaces, a handling circuit with inclines and programmable surfaces. The environmental conditions (temperature, pressure, humidity and weather) can all be controlled within rFactor Pro. The virtual proving ground includes a wide range of different roads and track surface sections.

3.3 Model build process

To compile the model for use in the vTag environment we first have to define all of the input and output signals at the top level of the model. We then utilize the Source Code Export feature of Dymola to compile the model using inline integration and export the model as c-code.

When Dymola exports the model in this way the code exposes a number of methods that enable the model equations to be coupled to a solver. As inline integration is used with the models the solver doesn't have to integrate any states but it does still have to handle events. The implementation of this solver has been optimized to run the model as efficiently as possible.

The interface between the physics model and the rest of the system is defined in additional c-files that are compiled with the solver and model equations to create the executable model. This interface contains a number of functions that are called by the system to handle the initialization of the model and the calculation of each time step. As the vision system will typically run at a lower frequency than the model the interface supports running multiple model steps each time the vision system asks for a step to be run.

For instance, this means the vision system can run at 500Hz but the model could run at 1000Hz or higher as appropriate. In this test system, the vision system is running at 400Hz and the model is running at 800Hz.

4 Conclusions

Using Modelica to define the vehicle model and Dymola to compile and export the model as c-code suitable for real-time simulation means that the physics model in a driving simulator can be very easily updated to test new concepts as well as explore setup variations of an existing design. The speed with which design ideas can be implemented in Dymola and compiled ready for the simulator means that it is possible for real drivers to start evaluating these ideas at a very early stage in the development process.

References

- [1] Otter M., Elmquist H., and Mattsson S.E.: The New Modelica MultiBody Library. Modelica 2003 Conference, Linkping, Sweden, pp. 311-330, Nov. 3-4, 2003.

- [2] Dempsey M. Picarelli A. Investigating the MultiBody Dynamics of the Complete Powertrain System. Como, Italy: Proceedings 7th Modelica Conference, 2009.
- [3] John J. Batteh Charles E. Newman. “*Detailed Simulation of Turbocharged Engines with Modelica*” Modelica Conference, 2008
- [4] Casella. F. et al. “*The Modelica Fluid and Media library for modeling of incompressible and compressible thermo-fluid pipe networks*” Modelica Conference, 2006
- [5] Christopher Depcik and Dennis Assanis “*A Universal Heat Transfer Correlation for Intake and Exhaust Flows in an Spark-Ignition Internal Combustion Engine*” SAE 2002-01-0372
- [6] Stodola, A. (1945). *Steam and Gas Turbines*. McGraw-Hill, New York. Reprinted by Peter Smith.
- [7] Eriksson, L. et al “*Modeling of a turbocharged SI engine*” Annual reviews in control 2002.
- [8] Heywood, B. *Internal Combustion Engine Fundamentals* McGraw-Hill
- [9] Robert Bosch Gmbh *Gasoline Engine Management* Bentley Publishers 2006.
- [10] Stone, R. *Introduction to internal Combustion Engines* SAE International 1999.
- [11] <http://www.mclarenelectronics.com/Products/Product/vTAG>
- [12] <http://www.logitech.com/en-gb/gaming/wheels/5184>
- [13] <http://www.rfactor-pro.com/>
- [14] <http://www.claytex.com/products/claytex-libraries/#vdlmotorsports-library>
- [15] <http://www.modelon.com/products/modelica-libraries/vehicle-dynamics-library/>
- [16] <http://www.mclarenelectronics.com/Products/Product/ATLAS>

Development of New Concept Vehicles Using Modelica and Expectation to Modelica from Automotive Industries

Yutaka Hirano

Toyota Motor Corporation, Future Project Division
1200 Mishuku, Susono, Shizuoka, 410-1193 JAPAN
yutaka@hirano.tec.toyota.co.jp

Abstract

Recently development of new-concept small vehicles for future mobility societies becomes very active. In this paper, development of simulation models of those new vehicles by Modelica is described. It became clear that such small vehicles tend to have reduced stability and handling ability than conventional vehicles. To cope with this problem, a benchmark study of designing vehicle control logic for an IWM (In-Wheel-Motor) vehicle was settled by Japanese society of automotive industries and academia. A brief description about this benchmark study is also given. At the end, requests to Modelica community from Japanese automotive industries are described.

Keywords: Future mobility vehicles; Stability and Handling Performance; Benchmark study

1 Introduction

To cope with future mobility society, development of many new concept vehicles is becoming increasingly active in recent years [1]. Those vehicles have characteristics of smaller size, lighter weight, less number of passengers than the conventional vehicles. Also those vehicles tend to be equipped with lower RRC (Rolling Resistance Coefficients) tires and new driving systems mainly using electric motors to achieve less emission and less energy consumption. Some of those future vehicles are equipped with IWM (In-Wheel-Motor) systems to achieve flexible layout of power-train and also advanced vehicle motion control [2]. Because such new-concept vehicles have different mechanical structure and control structure from those of conventional cars, it was necessary to make new models to estimate their motions by simulation. In this

paper, development of the simulation models of those new vehicles by Modelica is described. Those models were developed based on Vehicle Dynamics Library (VDL) of Dymola.

By the simulation, it became clear that such new small vehicles tend to have reduced stability and handling ability than conventional vehicles. To cope with this problem, a benchmark study of improving stability and handling ability of such new vehicles was settled by Japanese joint committee of automotive industries and academia. As a member of the committee, the author will introduce the benchmark study in this paper.

At the end of this paper, some requests from Japanese automotive industries to Modelica community are described. Those requests came from actual problem which was encountered by the users during the modeling and simulation works for new mobility vehicles.

2 Modeling and simulation of future vehicles

2.1 Target vehicles

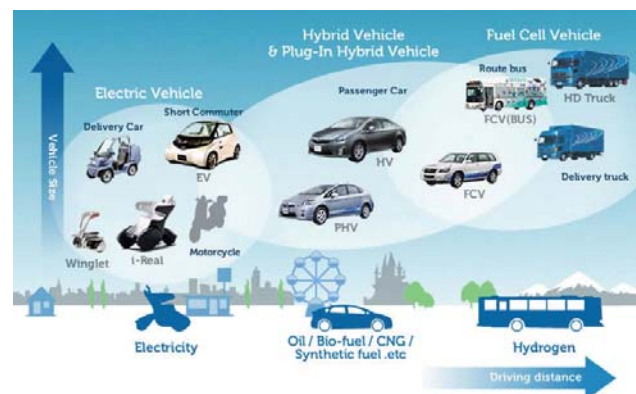


Figure 1: Toyota's scenario about future eco-cars

Figure 1 shows Toyota’s broad scenario about future eco-vehicles. As shown in the Figure, electric vehicles are thought suitable as future mobility for short distance. Those vehicles often have different structure from conventional cars. Thus it is necessary to make new models for new kinematics and control to simulate the motion of those new vehicles. In this paper, simulation models of a personal mobility ‘i-Real’ and a ‘short commuter’ by Modelica are described.

2.2 Simulation of a personal mobility ‘i-Real’

Figure 2 shows a photograph of Toyota’s proto-type personal mobility called ‘i-Real’. It has two front wheels and one rear wheel. Steering system is equipped with the rear wheel. The rear wheel is connected by a swing arm with the body and it is possible to change the length of wheel-base by controlling the angle of the swing arm actively. There are electric in-wheel-motors for each front wheel and rotation speed of each wheel can be controlled independently. Also there is a link to control the height of each front wheels independently. Thus, it is possible to control roll angle of the vehicle body against the ground actively.



Figure 2: Personal mobility ‘i-Real’

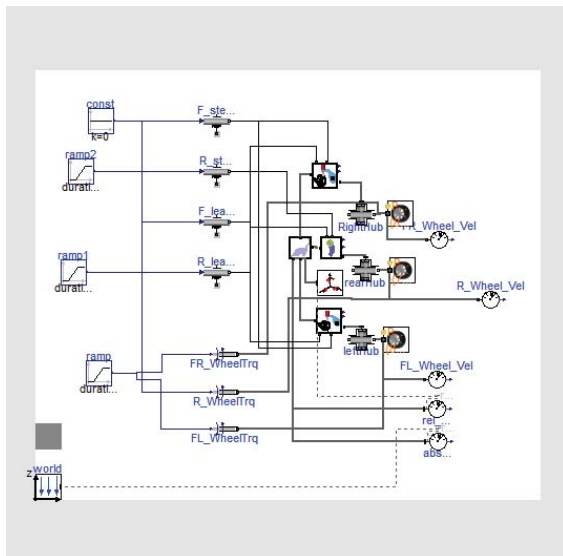


Figure 3: Dymola model of ‘i-Real’

Figure 3 shows Dymola model of the mechanical structure of ‘i-Real’. Each mechanical part is constructed by using Multi-Body-Systems (MBS) library and connected with the models of tires and environment of Vehicle Dynamics Library (VDL).

Figure 4 shows an animation result comparing a case when active control of wheel-base and roll angle was applied and a case when no control was applied while cornering. Figure 5 shows time plots of vehicle speed, lateral acceleration and yaw rate in this case. It was successful to simulate the effect of active roll-angle control and wheelbase control. Basic design of the vehicle motion controller was made upon this simulation model.

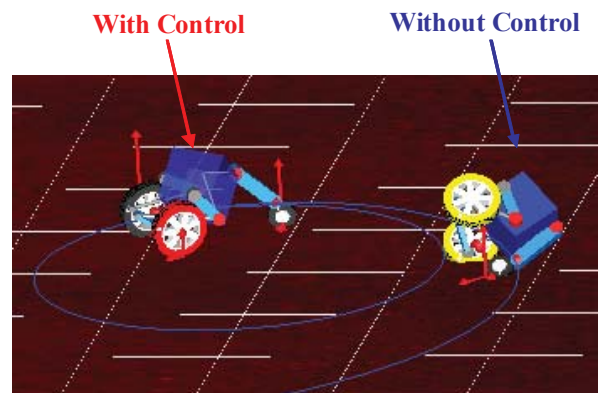


Figure 4: Simulation result animation of ‘i-Real’

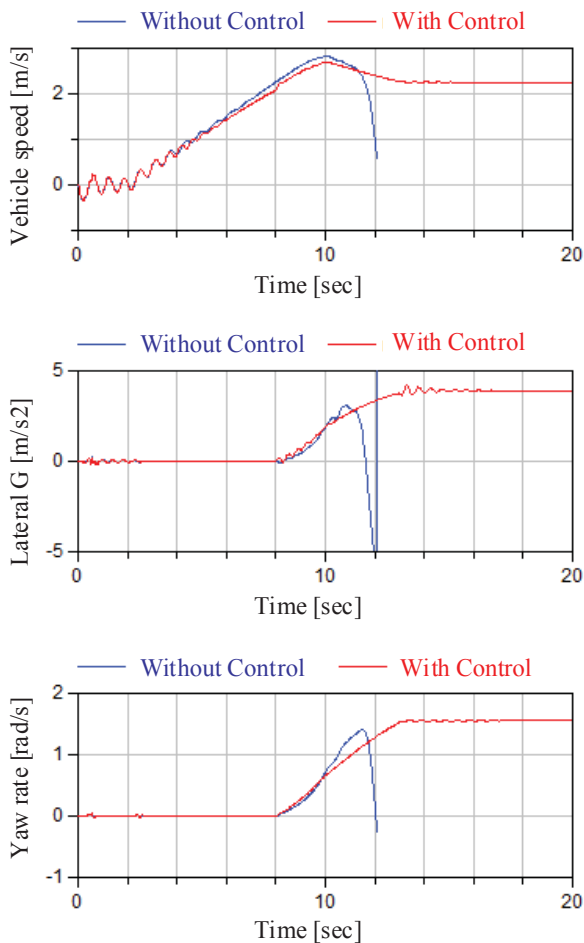


Figure 5: Time plots of 'i-Real' simulation

2.3 Simulation of a 'short commuter' vehicle

2.3.1 Background and purpose

Recently many small vehicles for short running distance mainly for the usage in a city area are proposed. Though, as shown in Figure6, lighter vehicle weight and smaller vehicle size tend to result in decreased resistance against external disturbances such as side-wind. Also tires having low RRC tend to have reduced side stiffness as compared to normal tires. Thus it is expected that handling performance of such small and light vehicles equipped with low RRC tires tend to be affected more than conventional vehicles.

To confirm this expectation, simulation of side wind test for both a conventional vehicle and a short commuter vehicle was executed. As an example of the short commuter vehicle, a small vehicle in which two passengers ride in series on the center of the vehicle was assumed. The specifications of both the short

commuter vehicle and a conventional vehicle are shown in Table 1.

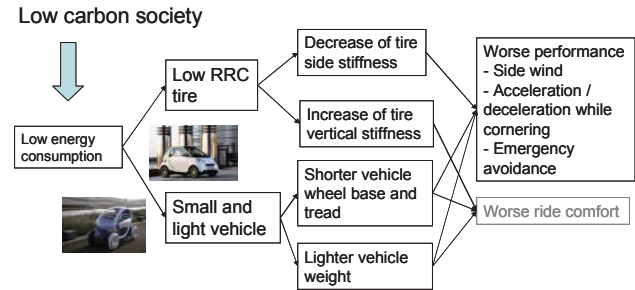


Figure 6: Problems for small commuter vehicles

Table 1: Specifications of vehicles

	Short commuter vehicle	Conventional vehicle
Weight	510 kg	1300 kg
Wheel Base	2000 mm	2600 mm
Width	1190 mm	1760 mm
Height	1460 mm	1515 mm

Vehicles run across a zone of side wind of 20m/s while running at 60km/h. Figure 7 shows the result of an animation for the open-loop side wind test, i.e. there is no control about steering. It is evident that the short commuter vehicle is affected a lot than the conventional vehicle by the side wind.

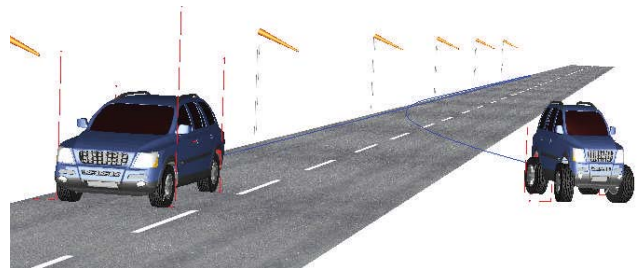


Figure 7: Result animation of side-wind test

Upon above backgrounds, it is planned to study about designing a control system for a future small IWM vehicle. The system enables control of individual steering angle and camber angle of each wheel as well as driving / braking torque of each wheel. The purpose of the study is to design a controller of an IWM vehicle to realize same level of handling and stability performance as conventional vehicles satisfy-

ing minimum energy consumption of IWMs simultaneously. This problem is announced to the wide area of academia as one of the benchmark studies from the automotive industries by joint committee of JSAE (Society of Automotive Engineers of Japan) and SICE (Society of Instrument and Control Engineers) about 'vehicle modeling and control research' in Japan. Anyone who wishes to join this benchmark study can freely obtain the model library from the web site below. The evaluation functions of the benchmark results will also be provided from the web site.

(http://cig.ees.kyushu-u.ac.jp/benchmark_JSAE_SICE/)

2.3.2 Structure of the simulation model

Figure 8 shows a whole structure of the simulation model based on VDL of Dymola. The model consists of a vehicle model and a driver model. The vehicle model includes 3D multi-body dynamics model of body and suspension. It is possible to control steering angle and camber angle of each wheel independently. The control of steering angle and camber angle of each wheel is realized by changing independently the length of two parallel lower arms of a double wishbone suspension which has an upper A-arm [3] as shown in Figure 8. Also a simple electric model of battery and IWM is included to calculate energy consumption of IWMs. There is a simple battery model which considers inner resistance and constant voltage generation. Electricity is provided to each DC motor and the motor converts electric current to driving / braking torque of each wheel by the following equation.

$$\tau_i = K_m \times i_i \quad (i = 1 \dots 4) \quad (1)$$

where τ_i : Motor torque, K_m : Constant, i_i : Motor current.

Power consumption of each motor is calculated by a multiplication of battery voltage and current flowing into the each motor.

There also is a model of driver's behavior which calculates commands for steering angle, acceleration pedal, braking pedal and so on. The driver model consists of function blocks of perception, planning and tracking respectively. The perception block calculates current vehicle status (position, speed, angle, etc.). The planning block settles target points on the path to be followed on the road from the information of the

perception block. The tracking block calculates driver's maneuver commands for steering, acceleration pedal, brake pedal and so on. These commands are transferred to the vehicle model to calculate the vehicle motion.

Finally all the models necessary for the simulation were integrated in one model library. Also test cases of desired tasks mentioned below were included in the library.

2.3.3 Description of desired tasks

The limitation of actuators of each wheel is shown in Table 2. There is no limitation for driving and braking torques of each IWM, but the requirement of minimizing energy consumption of IWMs is applied. The energy to control steering angle and camber angle of each wheel is not considered.

Table 2: limits of actuators for each wheel

Actuator	Limit
Steering angle	30 degrees (Front tires) 5 degrees (Rear tires)
Camber angle	10 degrees (All tires)

Four test scenarios were used for the benchmark study as below.

- 1) Acceleration while cornering on low friction road:
Accelerate the vehicle from initial speed 0[km/h] to 70[km/h] in 5 seconds on a slippery (coefficient of friction (μ) = 0.6) curve of R=50[m] as shown in Figure 9.
- 2) Deceleration while cornering on a sprit friction road:
Decelerate the vehicle from initial speed 70[km/h] to 0[km/h] in 5 seconds on a slippery split μ ($\mu = \{0.9, 0.4\}$) curve of R=50[m] as shown in Figure 10.
- 3) Double lane change:
Perform ISO double lane change task at the speed of 70[km/h].
- 4) Crossing side wind:
Run straight while crossing strong side wind at the speed of 70[km/h].

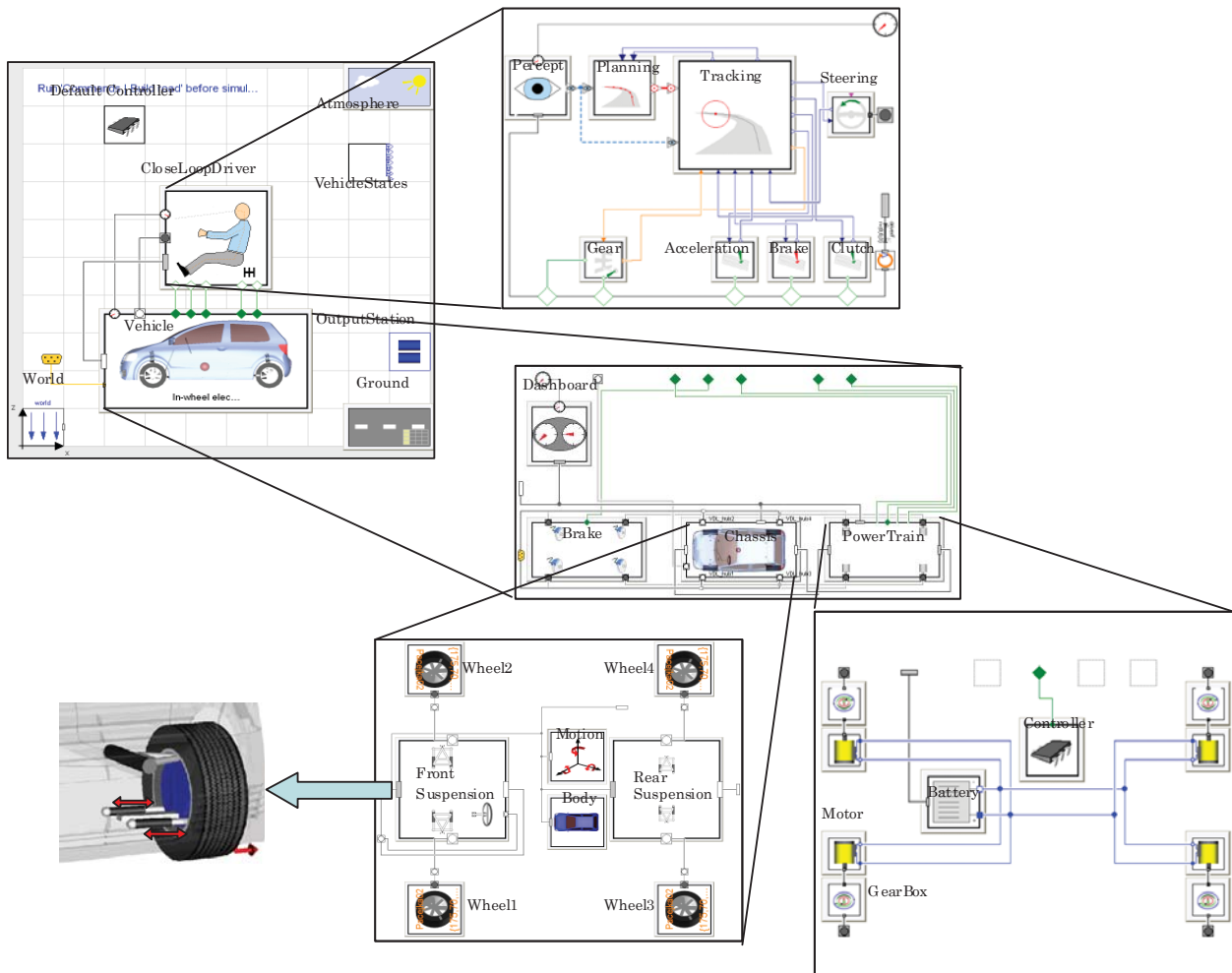


Figure 8: Diagram of main layers of the simulation model

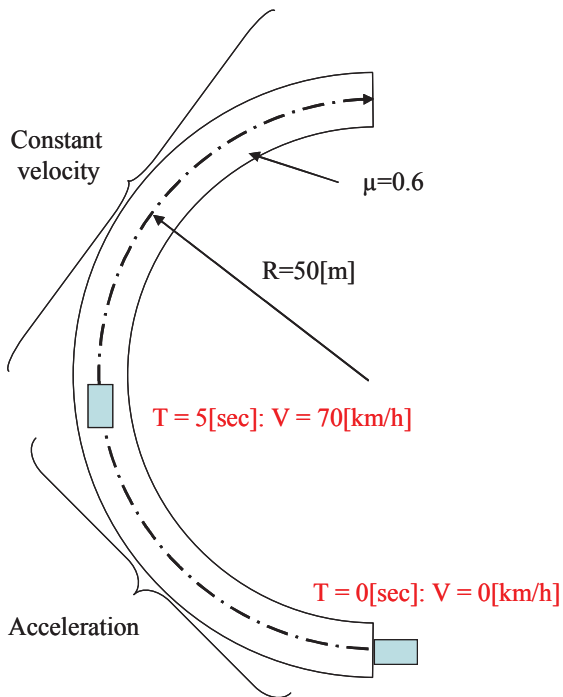


Figure 9: Test condition for 'acceleration while cornering' task

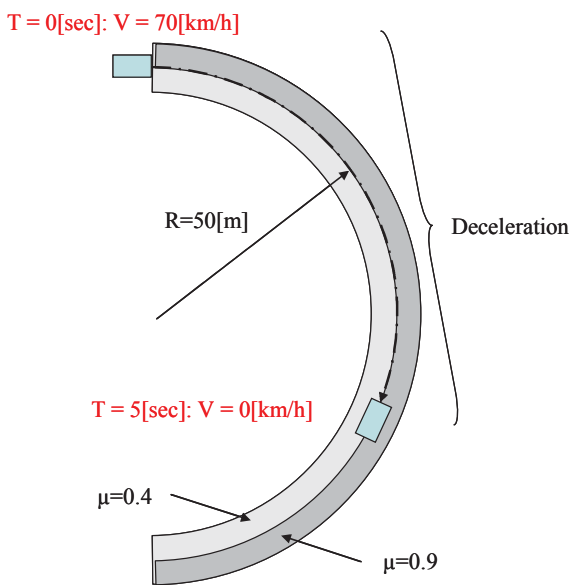


Figure 10: Test condition for 'deceleration while cornering' task

2.3.4 Vehicle model

As shown in Figure 8, the vehicle model consists of sub-models of brake, chassis and power train. Inside

the chassis model, multi-body dynamics of suspension links and joints are considered. Figure 11 shows Dymola model of the new suspension with two lower arms for which their length are actively controlled to control the camber angle and steering angle of the wheel independently. Also body motion is considered by multi-body dynamics model which has inputs from each suspension linkage. Because of this, the effects to body motion by suspension geometries such as anti-dive geometry, anti-squat geometry and so on can be considered. About tire model, 'magic formula model' (Pacejka'02) [4] is used.

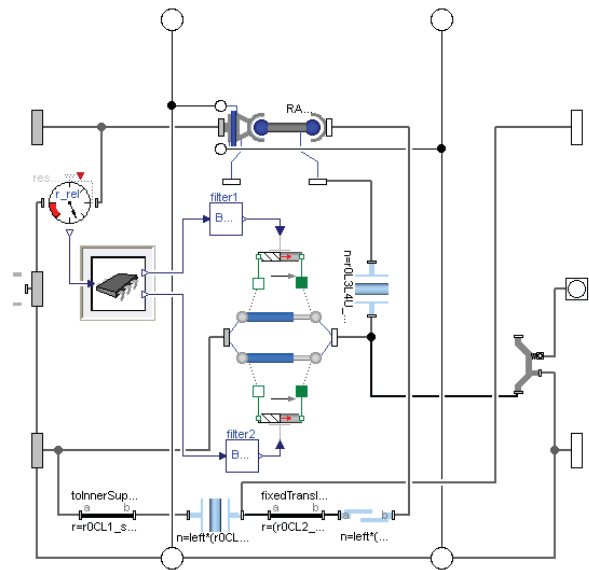


Figure 11: Dymola model of the new suspension

2.3.5 Driver model

Desired path and desired position of the vehicle on the path (target points) are settled on the road respectively according to the desired road shape and the vehicle speed profile for each task. 'Planning' block of the driver model shown in Figure 8 arranges target points along the desired path considering a preview distance of the driver model. 'Tracking' block calculates commands for steering angle, acceleration pedal angle and braking force respectively using the information from 'Planning' block and 'Perception' block. Each command is calculated as follows.

Steering angle command: str_cmd

$$str_cmd = \tan^{-1}\left(\frac{rV_x}{rV_y}\right) * str_gain + str_offset \quad (2)$$

where

rV_x : longitudinal distance along path between target point and current vehicle position,

rV_y : lateral distance along path between target point and current vehicle position,

str_gain : steering gain,

str_offset : offset value (optional).

Acceleration pedal command: acc_cmd

$$acc_cmd = K_{acc} \times D \quad (3)$$

and

Braking force command: brk_cmd

$$brk_cmd = -K_{brk} \times D \quad (4)$$

where

K_{acc} : Proportional gain for acceleration command,

K_{brk} : Proportional gain for braking command,

and

$$\begin{aligned} D = & K(vV_{P_x} - v_{veh}) \\ & + \frac{KNs}{T_d s + N}(vV_{P_x} - v_{veh}) \\ & + \frac{K}{T_i s}(vV_{P_x} - v_{veh}) \end{aligned} \quad (5)$$

Here,

K : Proportional feedback gain

T_d : Inverse of differential feedback gain

T_i : Inverse of integral feedback gain

N : Constant

s : Laplace operator

vV_{P_x} : Reference velocity along path

v_{veh} : Vehicle velocity along path

2.3.6 Controller model

To provide a template of controller model, an example model of the controller (default controller) is also provided in the model library. It is required for researchers of this benchmark study to propose revisions to the default controller (and also driver model if necessary) to realize the following demands.

- 1) Let vehicle yaw rate, side slip angle and lateral acceleration follow the desired values (ideal motion of conventional vehicle) and / or make a deviation from desired path to minimum under limitation of control amounts of steering angle and camber angle of each wheel.

- 2) Minimize energy consumption of IWMs.

The desired yaw rate, the desired slip angle and the desired lateral acceleration are calculated as bellows.

Desired slip angle:

$$\beta_{ref} = 0 \quad (6)$$

Desired yaw rate:

$$\gamma_{ref} = \frac{K_s}{1 + sT_s} \delta_{input} \quad (7)$$

Desired lateral acceleration:

$$G_{y_ref} = V \times \gamma_{ref} \quad (8)$$

Here, K_s and T_s are settled from the desirable motion of the conventional vehicle as follows.

$$K_s = \frac{(a_f + a_r)c_f c_r V}{a_r c_r M V^2 + a_f (a_f + a_r)c_f c_r} \quad (9)$$

$$T_s = \frac{M a_f V}{(a_f + a_r)c_f} \quad (10)$$

Here, following parameters are selected as a nominal value of the conventional vehicle for

a_f : Longitudinal distance between front wheel and CG (Centre of gravity)

a_r : Longitudinal distance between rear wheel and CG

c_f : Cornering stiffness of front two tyres

c_r : Cornering stiffness of rear two tyres

M : Mass of vehicle

V : Vehicle speed.

As a tentative example, the default controller calculates commands for the actuators as bellows.

Front steering angle:

$$\delta_i = str_cmd / G_s \quad (i=1, 2) \quad (11)$$

(G_s : Virtual steering gear ratio)

Rear steering angle:

$$\delta_i = 0 \quad (i=3, 4) \quad (12)$$

Camber angle of all wheels:

$$\alpha_i = 0 \quad (i=1\sim 4) \quad (13)$$

Driving / braking torque:

$$\begin{aligned} \tau_i = & K_{torque} \cdot (acc_cmd \quad \text{or} \quad brk_cmd) \\ & + K_p \cdot (V_{ref} - V) + K_i \cdot \int (V_{ref} - V) dt \end{aligned} \quad (i=1\sim 4) \quad (14)$$

where

K_{torque} : Constant

K_p : Proportional feedback gain

K_i : Integral feedback gain

V_{ref} : Desired vehicle speed

2.3.7 Tentative results of an example

As a tentative example, a result of applying the default controller in the case of ‘deceleration while cornering’ task is shown below. Figure 12 shows a time plot of vehicle speed. Only by above default controls it was not possible to trace the desired trajectory as shown in Figure 13. Figure 14 shows a time plot of side slip angle of both the cars in this case. Author now encourages many academic people to join this benchmark study.

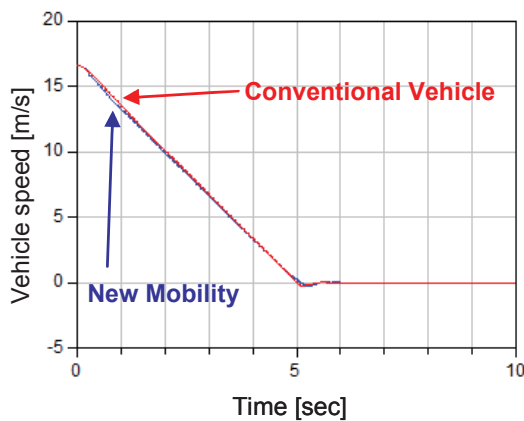


Figure 12: Vehicle speed of conventional vehicle and new mobility for deceleration while cornering’ task

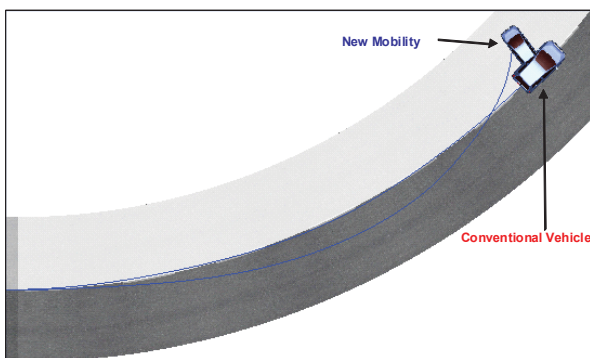


Figure 13: Trajectories of conventional vehicle and new mobility for ‘deceleration while cornering’ task

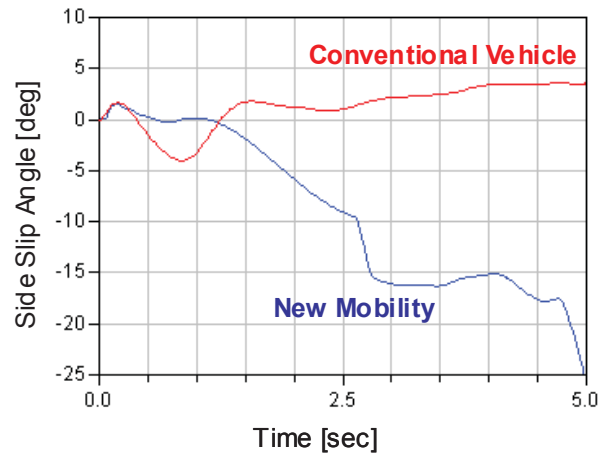


Figure 14: Vehicle slip angle of conventional vehicle and new mobility for ‘deceleration while cornering’ task

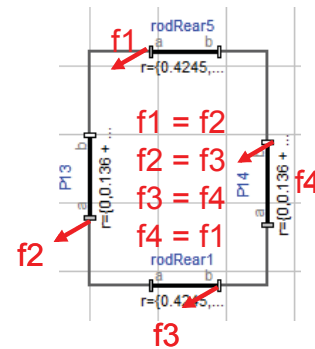


Figure 15: Limitation of calculating rigid mechanical loop of rigid elements by Modelica

3 Requests to Modelica from Japanese automotive industries

In the way of developing Dymola models for automotive applications, there occurred many requests to Modelica community from automotive industries. Table 3 summarizes the requests, though there are some ambiguous points and further discussion seems necessary. It is highly appreciated that Modelica Association will consider those requests in future development of Modelica specification, Modelica tools and also in future activity about Modelica. For this purpose, there is a high expectation to the activity of MIAB (Modelica Industrial Advisory Board).

As for the request number 21, the author will give an additional explanation. This request relates to a demand to convert CAD model to Modelica model di-

rectly. There often are cases of making a rigid structure by combining rigid elements when making mechanical structure models by CAD. However, by current limitation of Modelica, it is impossible to calculate such models because the force and torque acting on the every edge of the rigid elements are over constrained as shown in Figure 15. It is highly desired that such limitation will be removed in the future.

4 Conclusions

For some future mobility vehicles, Modelica models were developed for many virtual tests by the simulation. It was proved that such simulations were useful to estimate the motion of new mechanisms and also the effect of controls before making actual vehicles. To cope with the one of the potential problem of the future small-size vehicles, a benchmark study was proposed by Japanese committee of automotive industries and academia. It is highly welcome that many control researchers will join and challenge to the benchmark study. An organized session of this benchmark study will be held in coming IFAC-AAC (Advances in Automotive Control) 2013 symposium which will be held in September of 2013 in Japan.

Acknowledgement

Development of the model library for the benchmark study was done by close cooperation with Modelon AB.

References

- [1] Gombert B., "eCorner – the electric propulsion system of the future", Proceeding of Chassis Tech Plus 2011, pp. 803-813, 2011.
- [2] Katsuyama E., "Decoupled 3-D moment control by an In-Wheel Motor vehicle" Proceeding of Chassis Tech Plus 2011, pp. 133-150, 2010.
- [3] Andreasson J. : On Generic Road Vehicle Motion Modeling and Control, Doctoral thesis KTH (Royal Institute of Technology, Sweden) , Aeronautical and Vehicle Engineering, Trita-AVE, ISSN 1651-7660; 2006:85 ,2006.
- [4] Pacejka H.B., Tyre and Vehicle Dynamics, 2002, Butterworth-Heinemann, ISBN 0 7506 5121-5
- [5] Ito H., Ohata A., Butts K. : Equation-Based Model Data Structure for High Level Physical Modeling, Model Simplification and Modelica-Export, 4th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools. September, 2011, ETH Z"urich, Switzerland

Table 3: Requests from Japanese automotive industries to Modelica

Issue	No.	Requests
Readability of model	1	Improve readability of a model by avoiding difference between text-based description and GUI (connection editor) based description. (It's possible to write a model such that the parts are not connected in GUI but connected in text layer.) For example, making a guideline about the way of description for the definition of connection.
Code generation	2	Support C code generation for best-fit to compiler's optimization.
	3	Improve readability of generated C codes so that the modification by hand coding will be easier.
	4	Support C code generation for parallel processing. For example, specifying the importance of calculation causality between different physical domain and if the importance is small, then enable code generation so that the different domain can be calculated
Units	5	Support the unit of [rpm].
	6	Categorize and classify physical domain of SI units more clearly. (There are too many SI unit domains to search easily now.)
Libraries	7	Increase library blocks to connect different domains. (For example, between translational domain and rotational domain)
	8	Increase Modelica standard libraries
	9	Develop libraries for interaction of heat flow and mechanical systems (combustion engine, friction, damper, etc.)
	10	Develop libraries for interaction between mechanical vibration and sound field analysis.
	11	Develop libraries to simplify 3D flow analysis simulation to 1D flow
	12	To make commercial library independent for different modelica-based
Error handling	13	Improve the traceability of the reason of a error.
Usability	14	Make arbitrary one model class replaceable by simple GUI.
	15	Make it possible to specify physical variables to be shown in the simulation results (hopefully by simple GUI).
	16	Support revision management function for model classes and package files.
	17	Enable error handling and variables monitoring for protected models. (For example, models from suppliers.)
	18	Enhance FMI compatibility to other tools (Ex. GT-SUITES, CarMaker, CarSim, Star-CD, etc.)
Modeling methodology	19	Support of a new modeling methodology based on conservation laws of physical systems. (As HLMD from Toyota [5].)
	20	Model reduction: 1) Simplify precise equation-based physical models by numerical sensitivity analysis (ex. Sparse handling). 2) Identify parameters for non-linear dynamic parametric models from experimental data.
	21	Let the calculation for kinematic loops of rigid bodies possible.

A Modular Technique for Automotive System Simulation

Felix Günther, Georg Mallebrein, Heinz Ulbrich*

Robert Bosch GmbH, Stuttgart, Germany

*Technische Universität München, Institute of Applied Mechanics

{felix.guenther, georg.mallebrein}@de.bosch.com, *ulbrich@amm.mw.tum.de

Abstract

Increasingly challenging requirements such as environmental and safety legislation as well as increasing development costs are leading to a need for more overall system understanding in the automotive sector. Modelica, as a suitable way for multi-physics modeling, is therefore applied by Bosch, e.g. to investigate energy flows amongst domains.

We present a modular approach consisting of two parts to handle complexity and increase the performance: a modular library for the different domains and a co-simulation framework. To begin with, coupling aspects such as causality and communication are discussed in this context and their implementation is shown. A further focus is the variable macro step size that we developed within the framework for the automotive drive cycle simulation. The results of the modular approach are described and analyzed regarding error and performance aspects. Finally, challenges of the work are mentioned and an outlook, including FMI [2], [10], is given.

Keywords: co-simulation; automotive system simulation; multi-domain

1 Introduction

Scarcity of resources, legislation and customer demands continue to be the main driver for automotive manufacturers. New technologies such as hybridization or full electrification but also systems and components to increase the efficiency of the conventional powertrain help to reduce CO₂-emissions. Especially a supplier such as Bosch, providing a broad range of components as well as system solutions, requires a profound overall system understanding in all development stages. This is achieved by simulation, applying acceptably complex but comprehensive vehicle models. In contrast to signal-oriented or domain specific tools, Modelica proved to be a suitable way for physical, multi-domain and object-oriented modelling and is therefore applied, currently using

DYMOLA2012 [5] as simulation environment. Besides the physical domains, the vehicle controllers complete the models and hence a forward oriented, robust drive cycle system simulation is done. The resulting energy flows amongst the domains during a drive cycle give potential assessments of a certain system or component. This is compared to vehicle measurements (as e.g. in [16]).

A drawback of including all vehicle domains in one model is that the generated hybrid DAE system causes a high computational effort. Putting together the powertrain model with a detailed thermal and exhaust system model leads to simulation times several times slower than real-time on a Windows PC system.

To avoid this effect of complexity, the vehicle model is partitioned into subsystem models which are simulated in parallel using their locally adapted solvers. In a first approach, three subsystems were coupled, performing a co-simulation via TISC [17], at the expense of a precision loss but resulting in speed-up by factor of 5 on a single core PC. Additionally, using simulator coupling, the possibility to introduce existing MATLAB/SIMULINK or AMESIM models is given.

To reduce the numerical error introduced by partitioning and coupling, further development on coupling aspects was done. This was realized by applying MDPCosim [11] as master-slave co-simulation environment and expanding it with approximation methods and a variable macro step size control. In comparison to the approach with error estimation by repeating steps [4], a different approach with heuristic methods is being developed and in use for the presented drive cycle simulations.

2 Modular Simulation of Automotive Models

In order to optimize the energy consumption of a vehicle, simulation of the overall system is needed. This includes, besides the model of the powertrain with driver, engine, transmission, brakes and driving

resistances, all energy-relevant subsystems, namely the exhaust system, cooling and oil circuit as well as the electric power net and the system control. Accordingly, the models contain the mechanic, hydraulic, thermal, electric and boolean logic domain.

A modular realization was consequently chosen. On the one hand modularity is used to derive defined interfaces between the subsystems. This is obligatory for a collaboration of several developers or even several tools. On the other hand, a partition into modules for a possible co-simulation is prepared.

2.1 Motivation for modular simulation

One reason for modular simulation is to assemble models, developed in different tools. [7] shows different approaches, thus model coupling could be e.g. realized via FMI for model exchange [2]. Another reason is to partition the system to benefit of multi-rate time integration [1], [14], using different solvers for different dynamic behavior of subsystems. This can be used to meet real-time requirements for HiL simulation in the automotive domain, such as achieved in [9]. For HiL simulation fixed-step solvers are being used though, in contrast to the presented overall system simulation with variable DAE solvers.

In order to measure the computation time (t_{CPU}) overhead by assembling i subsystem models to the overall vehicle, a factor θ_A (1) is introduced.

$$\theta_A = \frac{t_{CPU, overall model}}{\sum_i t_{CPU, partial model i}} \quad (1)$$

During development of the subsystem models, e.g. the thermal system including the fluid circuits, the behavior of the vehicle is introduced by measured timetables and a standalone simulation is possible. Adding the thermal system model to the residual model of the vehicle containing already the other domains ($i = 2$), a θ_A of 12.1 was observed (for a more complex thermal model 19.2). Adding as an example a detailed model of the battery ($i = 3$) will again increase this factor. Therefore, instead of simulating the model with one solver, the modular approach is chosen, which additionally provides efficiency by simulating in parallel. Here, θ_A can be seen as an upper limit for the speed-up achievable by multi-rate advantages. For stability and accuracy reasons a partition leading to preferably weak coupling is useful, also giving the possibility to apply larger macro steps H . One possible partition method is described in [14]. Another method is the TLM (Transmission Line Modelling) approach, as presented in the Modelica context in [13]. TLM creates a modular

simulation by adding a solver to each component. Though, the advantage of symbolic manipulation of the equations for multiple components within one technical subsystem would disappear. Hence, in our work the nearby application along technical domain boundaries is chosen.

2.2 Coupling aspects

For a co-simulation of the modular vehicle model the implementation of different coupling aspects is necessary. In the following, those aspects are categorized and their application is described:

- **Synchronization:** different communication strategies between the solvers are possible. Figure 1 shows in its upper part a sequential asynchronous solution, e.g. described in [15] with advantages in accuracy and no necessity to define macro step sizes. The lower part presents the parallel synchronous solution, which is more efficient and therefore used here. The MDPCosim master controls the slaves including the models, who can run in parallel
- **Causality:** the advantage of Modelica with physical modeling and equation preprocessing disappears at the coupling interfaces, where causal, directed signals have to be used. MDPCosim covers the possibility for connecting slaves with coupling laws in the master [12] (flow-flow-coupling) and e.g. a reaction torque is retrieved in the master. For the applied step sizes and partitions in the vehicle simulation the more conducive technique of potential-flow-coupling is adopted. Details of the causal interface are commented in section 3.
- **Approximation:** the discretization introduced at the interface is adding an additional error to the simulation that can be reduced by approximation methods. Depending on the chosen synchronization scheme, different methods are possible: extrapolation, interpolation or even iterative such as described in [3]. It can be implemented in the master (constant), the slave (time varying) or both. On the present, parallel case, extrapolation including smoothing is chosen, see section 3.
- **Macro step size:** using synchronous coupling, a suitable macro step size has to be chosen. An efficiency gain for the overall simulation with acceptable co-simulation error needs to be combined. Therefore, investigation for fixed, predefined (timetable) and in conclusion variable macro step sizes was

done. As a result, a heuristic method was developed, which is described in detail in section 4.

- **Event handling:** occurring events in the subsystems will cause severe errors if the coupling values are affected, e.g. in a start-stop-strategy. This must be avoided thus no discontinuous signals are chosen in the present interfaces. Still, detecting and treating events during co-simulation is important for future work and one viable solution could be using FMI for co-simulation [10].

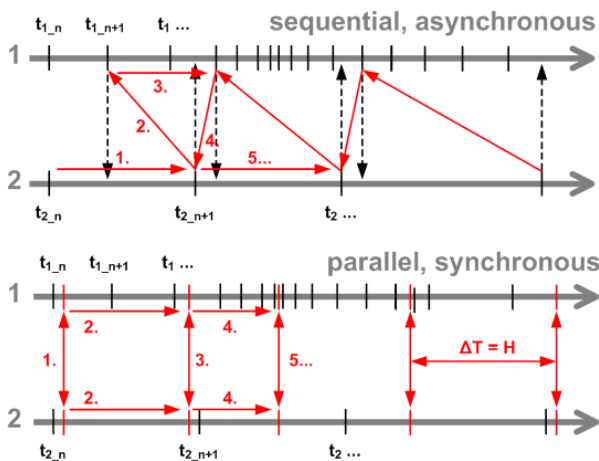


Figure 1: Two synchronization schemes (slaves 1, 2)

2.3 Evaluation of modular simulation

Illustrating the implementation of some coupling aspects and signal routing, figure 2 shows the implemented structure for the slaves in Modelica.

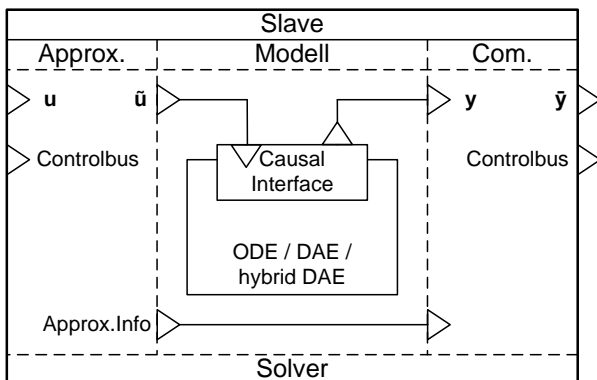


Figure 2: Structure of a slave model with signals

The input signals \mathbf{u} are extrapolated in the approximation section to $\tilde{\mathbf{u}}$. As wrapper to convert $\tilde{\mathbf{u}}$ and the output signals \mathbf{y} to the physical proper model the causal interface for different domains is modelled. The continuous \mathbf{y} then are communicated to the master as discrete signals $\tilde{\mathbf{y}}$. Additional, modular specific information is written to the control bus as explained later.

In order to evaluate the accuracy of the modular approach, a discretization error $\bar{\tau}_{CUM}$ is defined by (2).

$$\bar{\tau}_{CUM} = \frac{\int |\tilde{\mathbf{y}} - \mathbf{y}| dt}{t} \quad (2)$$

Notable at the physical interfaces this error is fed back and influences the behavior of \mathbf{y} , compared to the same states \mathbf{y}^* in a monolithic simulation (one solver). The correlation in the master between slave inputs \mathbf{u} and outputs \mathbf{y} is given as incidence matrix \mathbf{I} :

$$\mathbf{u} := \mathbf{I}\mathbf{y} \quad (3)$$

In order to take into account the accuracy augmentation by approximation, as well as the cumulated feedback influences, the co-simulation error

$$\tilde{\tau}_{CUM}^* = \frac{\int |\tilde{\mathbf{u}} - (\mathbf{I}\mathbf{y}^*)| dt}{t} \quad (4)$$

is introduced. In (3) and (5) t represents the simulated time.

Both accuracy and efficiency of the modular approach with co-simulation have to be regarded. Besides cumulating events and F-evaluations the speed-up factor S_{CS} (5) is important, having the reduction of simulation time of the overall vehicle model as motivation.

$$S_{CS} = \frac{t_{CPU,monolith.}}{t_{CPU,CoSim}} \quad (5)$$

The accuracy and efficiency of co-simulation was observed during the development of the library and the framework described below.

3 Implementation

As mentioned in the introduction, the modular automotive system simulation relies on two parts, the modular library and the co-simulation framework.

3.1 Modular library

The development of the library was based on using the Powertrain Library [6]. In addition, detailed models of the oil circuit, cooling circuit, HVAC, the exhaust system and the power net were developed. In order to enable configuring multiple classes of vehicles in different model granularity and combine them with existing in-house data libraries, the modular library was developed.

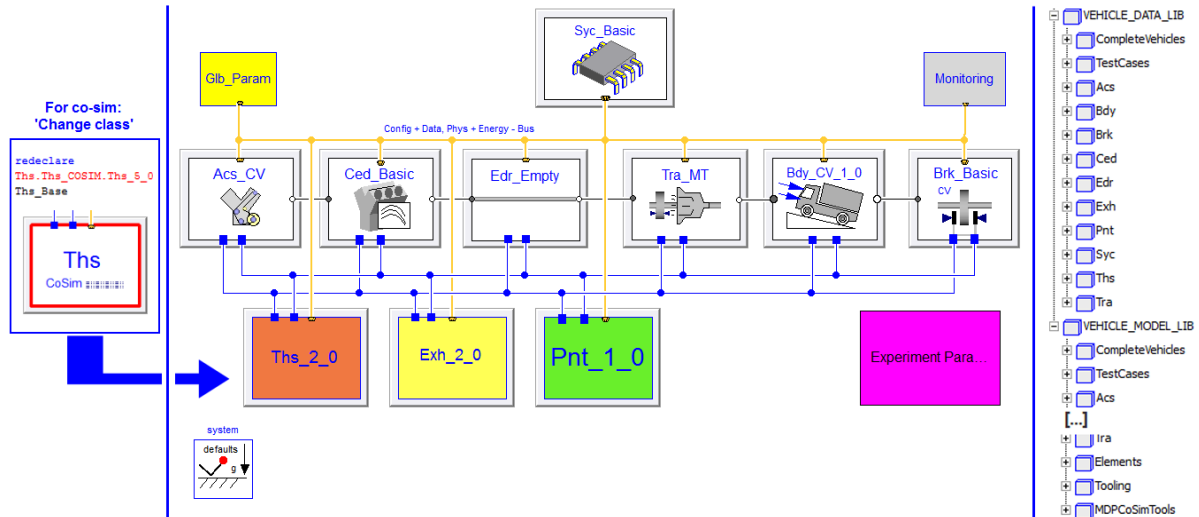


Figure 3: Example: overall vehicle model / data and model structure in the library

It contains a ‘_MODEL_LIB’ part for the model development and, mirroring the same structure, a separate ‘_DATA_LIB’ part, where vehicles and subsystems are configured, parameterized and set up for different (drive cycle) simulation experiments. Figure 3 shows a top-level model of a commercial vehicle. On the right part, the library structure with model and data part is shown. The different subsystems, e.g. for combustion engines (‘Ced’) or thermal systems (‘Ths’), can be changed by redeclaring the class with other data lib models. In the same way, a subsystem can be set up to be co-simulated, as shown in the left part of figure 3: The Ths-model is replaced by an interface (‘Ths CoSim’) directing to the thermal system co-simulation slave, which can be found as standalone model in the library structure and will be run in parallel.

Modularity is also represented in the subsystem models. Figure 4 depicts the thermal system. It contains replaceable models for the energy balance, combustion engine heat, cooling circuit, oil circuit and HVAC.

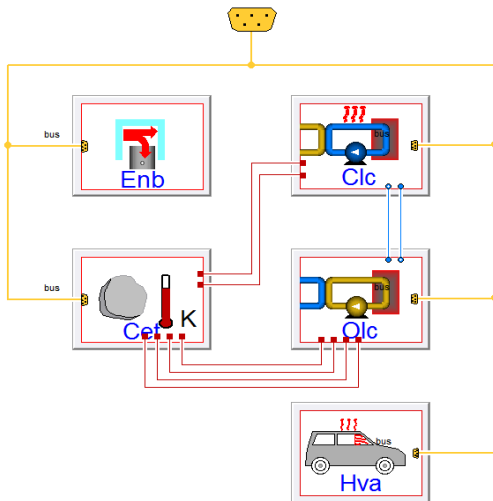


Figure 4: The thermal system model

In order to allow maximum modularity accompanied by physical coupling between the subsystems, causal interface models are introduced for different domains enabling signal exchange with a ‘causalSubBus’. In such manner e.g. the oil pump is coupled to the powertrain part. The related flange interface in figure 5 shows the crank part.

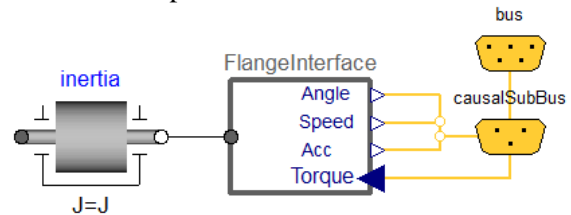


Figure 5: Causal interface for flanges

Depending on the macro step sizes, a flow-flow or a potential-flow-coupling can be chosen. Similar interfaces are used for the thermal part. For communication with MDPCosim and signal approximation, a configurable interface model is in the library, figure 6.

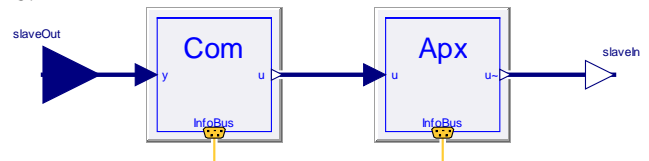


Figure 6: Co-simulation interface model

Different slave approximation methods, partly in combination with master approximation of flow variables, have been tested on a two-mass-oscillator model as well as in the vehicle context. Following, different methods, such as Taylor, Lagrange and Hermite polynomials and a transition method, smoothing signal jumps, are implemented for the library and applied in the ‘Apx’ block. All methods can handle a variable macro step size. Additionally,

an ‘infoBus’ is introduced containing information about the signals approximation.

3.2 MDPCosim framework

The latter information can be fed via a control bus and used for master algorithms. This architecture is described in figure 7. It shows the configuration of the MDPCosim framework [12] and its adaption as vehicle co-simulation environment in C++. The abovementioned overall vehicle model is represented as slave 1 ... N . An overhead process actuates the master and slave processes. These run in parallel, while the co-simulation is controlled by the master. This includes synchronization, connecting the signals (feedthrough or coupling law with approximation [12]) and the macro step size algorithm.

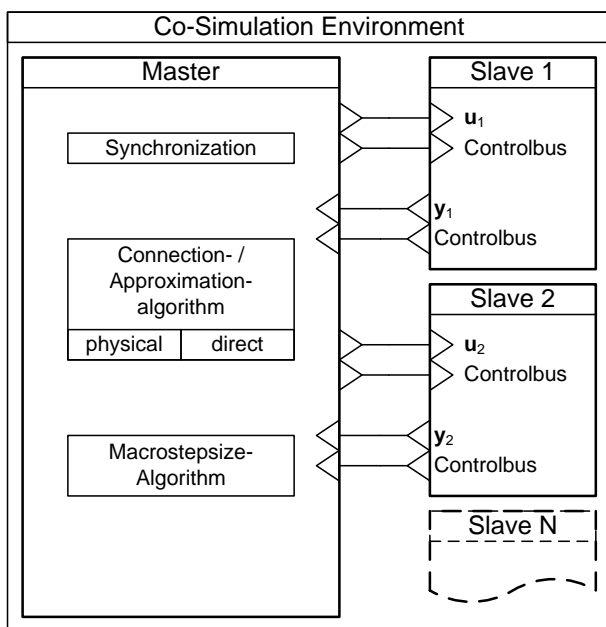


Figure 7: Architecture of the adapted environment MDPCosim

As inter process communication, shared memory is used; TCP/IP is planned for future work. Besides the coupling signals (u , y) the control bus connects master and slaves. It contains: derivatives of y , step size H , T_{next} , \vec{t}_{CUM} and information about approximation, local step sizes h and events. This information is handled within the master algorithms.

The inner layout of the slaves is shown in figure 2 and figure 6 respectively.

3.3 Batch co-simulation

As a tool for the development of modular methods, MDPCosim was expanded with a superimposed algorithm that allows automated batch runs. This is used to sweep parameters of the master as well as the

slave models. Hence, fitting of model parameters is possible or a variety of co-simulations needed for requirements engineering of a certain component in the overall system context can be run.

Figure 8 demonstrates the data and process flow of a batch run. It is configured, using a file that contains the following information: number of runs, type, identifier of a parameter (file), (min and max) values. The different types are ‘variants’, ‘parameter’, and ‘autoParaVari’. The type ‘variants’ is followed by a file identifier defining numbered versions of a model or master parameter file or different model files. The other types allow naming a parameter to be varied, giving all values or giving a minimal and maximal value.

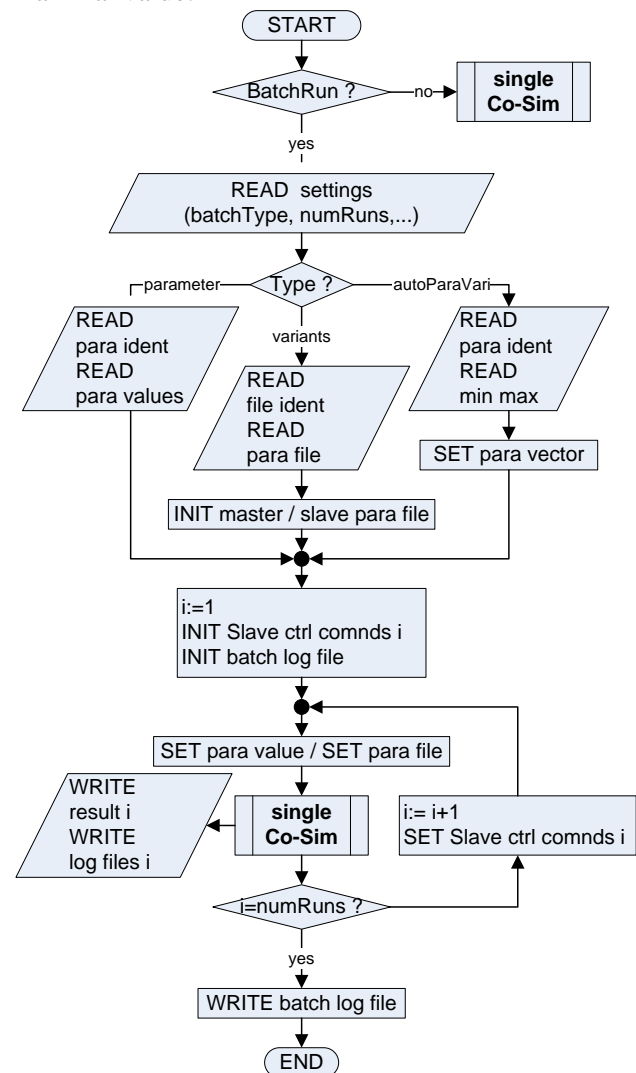


Figure 8: Flow chart of the batch architecture

The result files and a batch log file can be imported in MATLAB to be commonly evaluated for the aspects described in section 2.3. By means of this evaluation the development of approximation and macro step size algorithms is done.

4 Macro Step Size Control for Drive Cycle Simulation

The objective of modular automotive simulation is to increase simulation speed, while keeping the co-simulation error acceptable. In contrast to the sequential asynchronous approach, see 2.2, for the parallel synchronous technique, defining the macro step size H is necessary. Setting $H = h_i$ (with h_i : local step size of slave i) is not conducive, if we consider the effort for each macro step with an event in all slaves, waiting for synchronization and the master algorithm for coupling. Thus relatively large H are pursued. To reduce the following discretization error, the extrapolation methods are adopted and an algorithm for variable H is developed.

State of the art of simulation tools and the abovementioned context (section 2 and 3) lead to the following boundary conditions for the macro step size control: large subsystem models are solved with a commercial simulation tool and are therefore seen as black boxes for the master; overall, the coupling is kept weak (slow changing temperatures in the thermal model / small pump inertia compared to the powertrain inertia.); the drive cycles (e.g. [8] or [18]) span more than 1000 s and macro steps in the range of 0.1 s and 10 s are chosen; the drive cycles provide an approximate predictive behaviour of the overall vehicle. Embedded methods or the Richardson method such as presented in [4] are neither conducive (for efficiency reasons) in the present use-case nor possible (yet), since a macro step cannot be repeated. Thus, the methods, described in this paper are heuristic and strictly monotone procedures based on indicators.

The chosen approaches go without the need to repeat steps and partly establish the general correlation:

$$H = H(y, \dot{y}, h, \bar{\tau}, \tilde{u}(t), t_event, \dots, p, \dots) \quad (6)$$

With p as parameters to be defined by the user and $\tilde{u}(t)$ derived by the knowledge about the used approximation method for each signal. Based on (6) different approaches can be combined:

- H is determined basing on additional user input parameters p or simply the common RTOL/ATOL user limits.
- H is determined by one leading slave output y , by multiple or all outputs of all slaves y .
- H is determined with local slave information about derivatives \dot{y} , error $\bar{\tau}$, local step size(s) h , approximation and events. This information is provided via the control bus (Figure 2, 7).

- H is determined with or without quasi-error estimation based on $\bar{\tau}$ and \tilde{u} .

In figure 9 the master algorithm with the step size control ('H algorithm') is explained. If the parameter for H is set to <0 , a table file with $H = f(t)$ is used and if H is set to 0 the H controlling algorithm is initiated. After initializing the slaves and the master including the H algorithm initialization, the co-simulation cycle starts. Within each cycle, after executing each macro step, the H algorithm is called and can set a new value for H .

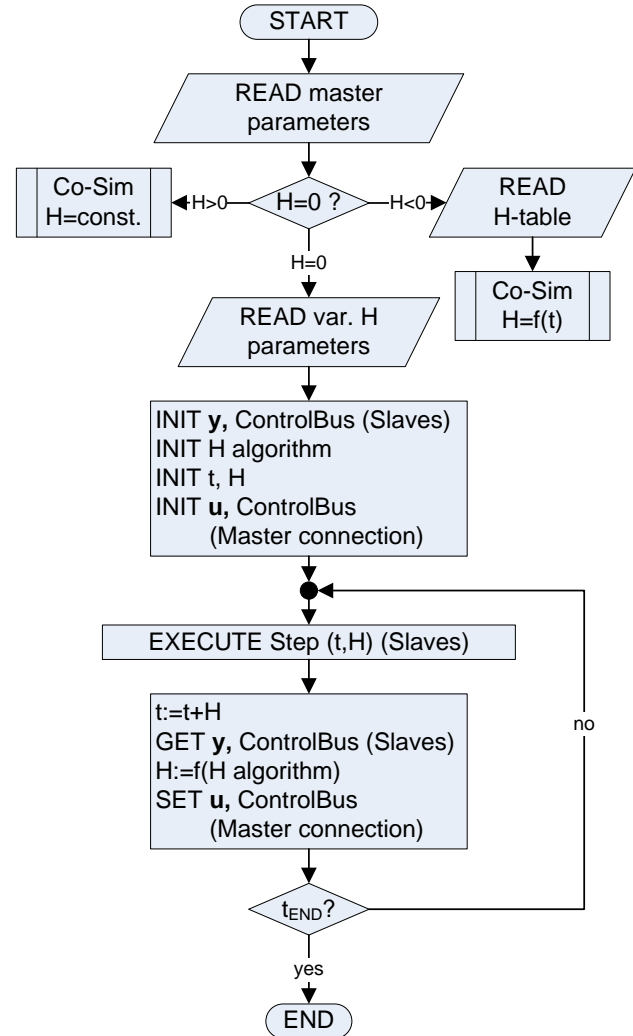


Figure 9: Flow chart of the master algorithm

In figure 10 an example of a master parameter file is shown. Starting with the entry for the co-simulation end time, the second line defines H . If it is 0, the algorithm continues reading the parameter file with a line for the chosen H algorithm type and a line for start value for H (optional), followed by type specific parameters (see next section).

```

1180
0
H=f(singleSlave,scalarY,parameters)_1b
0
0 8
0.4 2 2
    
```

Figure 10: Example: Master parameter file for variable macro step size

Currently, the development of the more sophisticated H algorithms is still ongoing. However, first algorithms are implemented and in use. A state-lead algorithm is described in 4.1.

4.1 Implementation

In a drive cycle, the desired vehicle speed v and the gear is given as $v = f(t)$ and $\text{gear} = f(t)$. Additionally, the resulting kinematic states in the powertrain dominate the overall vehicle model behavior. Accordingly, the macro step sizes are based on gradients of the vehicle speed (\dot{v}) or the engine speed ($\dot{\omega}_{eng}$) as leading states and indicator for most changing rates of the model states. Thus the macro step size H is set inversely proportional to the last gradient value.

As shown the example in figure 10, line 5, the user has to provide parameters for the index number of the slave i and the belonging index number j of the leading state. This has to be completed with the last line of parameters with values for H_{min} , H_{max} , method tuning parameters and minimum and maximum gradient values. To reduce the user input, a method without the latter entries was developed. The correlation follows with

$$H = H_{min} + [1 - \min(1, \dot{y}_{dl})]^{p_0} (H_{max} - H_{min}) \quad (7)$$

with a dimensionless \dot{y}_{dl} :

$$\dot{y}_{dl} = \frac{|\dot{y}_{i,j}|}{p_1 \dot{y}_{avg}} \quad (8)$$

depending on a weighted mean value over the current simulation time. The algorithm can be optimized with the remaining parameters: H_{min} , H_{max} , p_0 , p_1 . For this purpose, batch runs for each parameter are done with representing use case models and evaluated using MATLAB. One of the results is shown in figure 11, where a variation of H_{min} from 0.05 s to 0.5 s, holding all other master parameters constant, is presented. It covers the evaluation for number of steps, speed-up S_{cs} (5) and the error $\tilde{\tau}_{CUM}^*$ (4). In that manner, the parameters were optimized for a certain drive cycle.

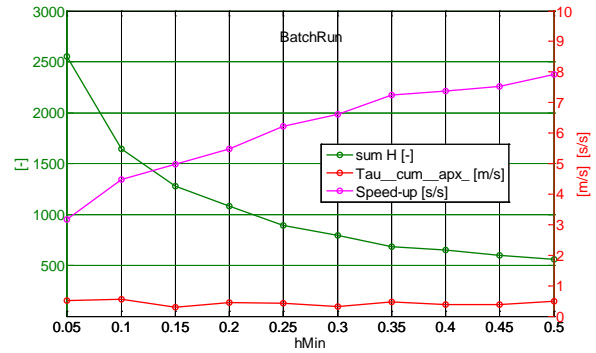


Figure 11: Evaluated batch run varying H_{min}

The macro step sizes during a drive cycle (NEDC) in figure 12 are in the range of 0.4 s and 2 s and result in an average step size of 0.92 s.

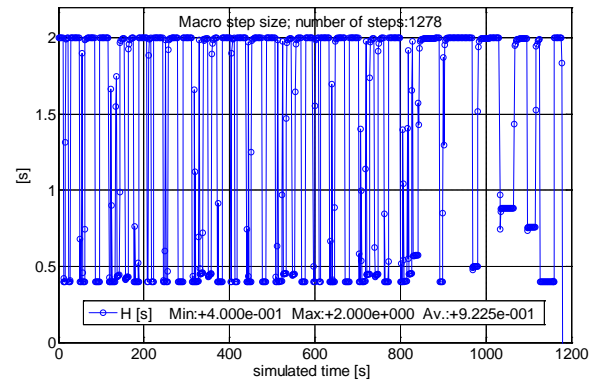


Figure 12: macro step sizes in a drive cycle

To evaluate the algorithm, this is compared to a fixed H co-simulation with this average value. The comparison is shown in table 1.

Table 1: macro step size control method evaluation

	variable H	fixed H (0.92 s)
S_{cs} [s/s]	4.516	4.502
$\tilde{\tau}_{CUM}$ [m/s]	0.036	0.060

With a comparable speed-up factor the cumulated discretization error could be significantly reduced. With adapted parameters, the method was also successfully applied for the two-mass-oscillator test case. However, the user needs knowledge about the model and the coupling method. Therefore ongoing investigation is done on methods with less user inputs on the one hand and embedding more local information as well as a control strategy for approximations on the other hand.

5 Use-Cases and Results

All models presented in this paper are simulated using DYMOLA [5] and co-simulation is done by coupling several DYMOLA processes. The vehicle hybrid DAE models require the use of the ‘dassl’ solver and the same integration settings are used for all experiments. The modular simulation was investigated on test use-cases, a two mass oscillator (TMO) and simple thermal modal as well as in the overall automotive context. For completeness, some results of the approximation tests with the TMO are given in the following.

5.1 Two-mass-oscillator test case

To develop the causal flange interface, an undamped rotating TMO is modeled with high frequency of the left and low frequency in the right mass and simulated, using MDPCosim. Thus the direction and the combination of approximations can be distinguished. Table 2 shows some results. The co-simulation error after 20 s with $H = 5$ ms is compared for potential-flow-coupling (angle φ_L to the right side) and flow-flow-coupling with different approximation methods. It could be reduced by more than two orders.

Table 2: TMO: improvement with approximation

	$\tilde{\tau}_{\text{CUM},\varphi_L}^*$ [rad]
potential-flow-coupling: no approx. (0. order extrapolation)	4.43e-1
potential-flow-coupling: phi_left: first order transition tau_right: 4-point-lagrange	2.90e-3
flow-flow-coupling: master: 2 nd order method slaves: first order transition	1.38e-3

5.2 NEDC: vehicle with detailed thermal system

Following the conditions for overall vehicle simulation with larger step sizes, in the current state of the modular library only potential-flow-coupling and signals without direct physical reaction are used, such as the fuel mass flow. For temperature signals a 4-point-lagrange polynomial and for (rotational) speed signals a first order taylor or the transition method is configured.

Here, as an example a conventional passenger car model is coupled with a detailed thermal system model, similar to the one in figure 4, but with only a two-thermal-mass motor block model and simplified models of the cooling system and HVAC. The results are evaluated by referencing the same overall vehicle model, simulated without co-simulation with

only one solver. For the NEDC (simulated time 1180 s), the computing time was 1400 s. With approximations and the variable macro step size (see 4.1; $H: \emptyset 0.92$ s) the co-simulation computing time was 310 s (speed-up: 4.5 and real-time capable). It lead to an acceptable error e.g. of the fuel consumption value of $\ll 1\%$. Compared to $\theta_A = 12.1$ (see 2.1) there is more speed-up capability. This can be reached using larger $\emptyset H$, however finally leading to unacceptable accuracy. For more complex models of the thermal, exhaust system or the powernet, more speed-up is reached.

Figure 13 shows three different simulations for the same acceleration sequence in this cycle. The simulation results for the engine speed of the reference, a co-simulation with correlating fixed H and the co-simulation with variable H and 0. order extrapolation are compared.

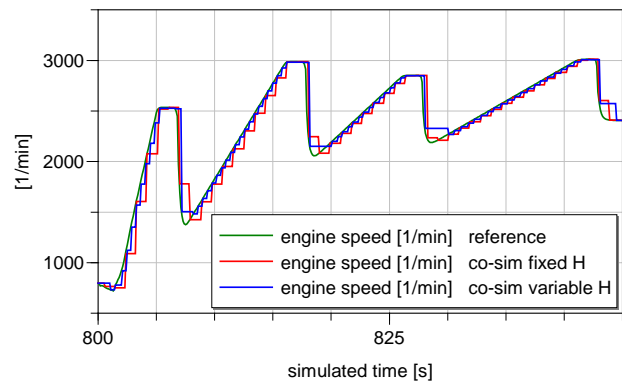


Figure 13: engine speed in an acceleration sequence

The same three simulation results as in the upper figure are taken in figure 14. To compare the behavior of an approximation method in combination with variable macro steps additionally, the warming-up curves of the average oil temperature are taken.

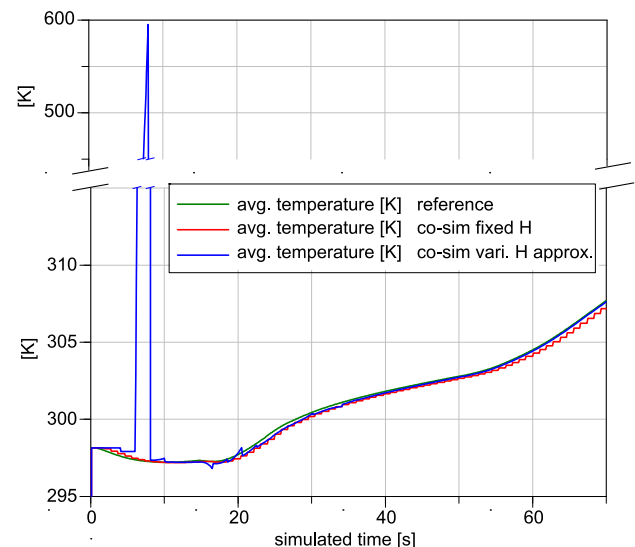


Figure 14: warming-up at cycle start / negative approximation effect.

The approximated curve is mostly more congruent to the reference. However, in the first part, the problems of applying higher order extrapolation together with large step sizes is obvious.

This is one of the challenges in using a parallel and strictly monotone modular technique. Therefore, as mentioned in 4.1, the control bus has to be adopted for a quasi approximation order control together with further improved step size control methods.

6 Conclusions and Outlook

In this paper we presented a Modelica based modular approach for overall vehicle system simulation. The advantages of using co-simulation in this context are deduced and achieved computational speed-up results are shown. The modular approach consists of two parts: a modular multi-domain vehicle library and the adapted co-simulation framework MDPCosim [11]. The modular library allows configuring complete vehicles by assembling the needed subsystem models, which is also possible as co-simulation slave to be simulated in parallel. Additionally, it provides interface models that can be easily configured by the user to set up a co-simulation run. The implementation of different categorized coupling aspects is shown. In particular, a heuristic method for macro step size control that is used for the overall vehicle simulation is explained. As advantage its parameters were chosen according to the a priori known drive cycle. Though, there are many challenges, which have to be regarded to make modular simulation more applicable.

Thus, there is remaining work to be done. A preferable way is the adoption of FMI [10], once a reliable implementation also for the mentioned large multi-domain models is available (not the case at the beginning of the present work). The FMI standard provides a suitable set-up for the algorithms described above. Consequently an even more common use of the modular library approach will be feasible, also including more different tools.

Acknowledgements

We are grateful for the collaboration with TU München providing access to MDPCosim; Marcus Schulz, Cooperative State University Stuttgart, for fruitful discussions and TLK Thermo GmbH for kindly providing TISC [17]

References

- [1] Arnold, M. Multi-Rate Time Integration for Large Scale Multibody System Models. IUTAM Symposium on Multiscale Problems in Multibody System Contacts, Stuttgart, Germany, 2006.
- [2] Blochwitz, T. et. al. The Functional Mockup Interface for Tool independent Exchange of Simulation Models. Proc. 8th International Modelica Conference. The Modelica Association. Dresden, Germany, 2011.
- [3] Busch, M., Schweizer, B. Numerical Stability and Accuracy of Different Co-Simulation Techniques: Analytical Investigations Based on a 2-DOF Test-Model. The 1st Joint International Conference on Multibody System Dynamics, Lappeenranta, Finland, 2010.
- [4] Clauß, C., Arnold, M., Schierz, T., Bastian, J. Master zur Simulatorkopplung via FMI. ASIM-Konferenz STS/GMMS 2012, ISBN 978-3-901608-39-1, Wolfenbüttel, Germany, 2012.
- [5] Dassault Systèmes, Dymola 2012, 2011 URL <http://www.3ds.com/products/catia/portfolio/dymola>
- [6] DLR Institute of Robotics and Mechatronics, The Powertrain Library (Version 2.1.0), 2011 URL http://www.dlr.de/rm/en/desktopdefault.aspx/tabid-5312/8907_read-16072/
- [7] Dronka, S. Die Simulation gekoppelter Mehrkörper- und Hydraulikmodelle mit Erweiterung für Echtzeitsimulation. Dresden, Germany, PhD thesis, Technische Universität Dresden, 2004
- [8] European Commission. NEDC. Consolidated Directive 70/220/EEC, 2006.
- [9] Faure, C. et al. Methods for real-time simulation of Cyber-Physical Systems: application to automotive domain. Proc. 1st IEEE Workshop on Design, Modeling and Evaluation of Cyber Physical Systems, Istanbul, Turkey, 2011.
- [10] FMI Specification 2.0 Beta 3, available for free from URL <http://www.functional-mockup-interface.org/> (2.0 beta)
- [11] Friedrich, M. Parallel Co-Simulation for Mechatronic Systems. München, Germany: PhD thesis, Technische Universität München, Institute of Applied Mechanics, 2011.

- [12] Friedrich, M., Schneider, M., Ulbrich, H. A Parallel Co-Simulation for Mechatronic Systems. The 1st Joint International Conference on Multibody System Dynamics, Lappeenranta, Finland, 2010.
- [13] Johansson, B., Krus, P. Modelica in a Distributed Environment Using Transmission Line Modelling. Proc. Modelica Workshop 2000, Lund, Sweden, 2000.
- [14] Kanth, D. Zur steifigkeits- und kopplungs-basierten Partitionierung mechatronischer Systeme. Stuttgart, Germany, PhD thesis, University of Stuttgart, 2010
- [15] Petridis, K., Klein, A., Beitelschmidt, M. Asynchronous method for the coupled simulation of mechatronic systems. Proceedings in Applied Mathematics and Mechanics, Bremen, Germany, 2008.
- [16] Rumbolz, P., Baumann, G., Reuss, H-C. Messung der fahrzeuginternen Leistungsfluesse im Realverkehr. ATZ 05 2011, Germany, 2011
- [17] TLK-Thermo GmbH, TISC, 2012, URL <http://www.tlk-thermo.com>.
- [18] UNECE, WLTC v4, 2012, URL <http://www.unece.org/>.

Modeling Vehicle Drivability with Modelica and the Vehicle Dynamics Library

John Griffin¹

John Batteh²

Johan Andreasson¹

¹Modelon AB

²Modelon, Inc.

Ideon Science Park

Ann Arbor, MI

Lund Sweden

United States

john.griffin@modelon.com john.batteh@modelon.com johan.andreasson@modelon.com

Abstract

This paper highlights the use of multi-domain physical models for simulation of vehicle drivability applications. The models are implemented using the Vehicle Dynamics Library and Engine Dynamics Library from Modelon. The application examples include vehicle launch, vehicle start-stop, and transmission shift events. The examples are structured to illustrate how increasingly sophisticated models provide additional model fidelity or increase the drivability phenomena observed.

Keywords: vehicle dynamics; drivability; vehicle modeling; powertrain; engine; transmission; launch; NVH

1 Introduction

To meet increasingly stringent fuel economy and emissions standards, automotive original equipment manufacturers (OEMs) and suppliers have sought novel technologies to meet customer demand constrained by the regulatory environment. As system complexity increases, the need for increasingly sophisticated analytic tools to perform concept evaluation, capture multi-domain system interactions, and develop and validate control strategies grows. Modelica has been used extensively in the automotive community for modeling and simulation of vehicle dynamics and handling [1], transient engine modeling and performance [2] [3], vehicle thermal management [4], air conditioning systems [5], and vehicle fuel economy and emissions [6].

While customers demand continued refinement in vehicle performance attributes, they also demand no compromises in vehicle comfort and vehicle drivability. Furthermore, many system design or control actions improve one attribute potentially at the expense of another or several others, typically drivability or comfort. Automotive manufacturers are acutely

ly aware of the market requirements to achieve best in class levels of vehicle performance and drivability. For example, customers may report drivability related issues such as shift busyness resulting from an increased number of vehicle shifts to optimize fuel economy for transmissions with more gears. Shift performance and feel are also common customer complaints. With increasing use of start-stop technology (see Figure 1), customers experience many more starting events, and their expectations regarding these events differ in driving mode as compared with a single start in park in a garage or parking lot. Vehicle launch with both conventional and especially with start-stop technology can be especially problematic from both a performance and drivability standpoint. With multiple power paths in both conventional and hybrid vehicles, interactions between subsystems can lead to vehicle vibrations typically felt at the seat track by the customer. With the accelerated adoption of dual clutch transmissions, driveline vibrations induced by clutch dynamics are becoming a drivability concern. Variations in clutch friction material, alignment, etc. can affect both nominal performance and drivability, and data to characterize the key components is often not available or considered proprietary by the suppliers.

This paper describes several different vehicle drivability applications. These models are implemented using components from the Modelon Vehicle Dynamics Library (VDL) [1] and Engine Dynamics Library (EDL) [8]. These examples highlight the multi-domain approach needed to simulate vehicle drivability issues. The examples are also structured to illustrate how increasingly sophisticated models provide additional model fidelity or increase the physical phenomena observed. The sample applications in this paper include vehicle launch, vehicle start-stop, and transmission shift events. The applications also include different modeling approaches for the engine with both a conventional automatic and dual clutch transmission.

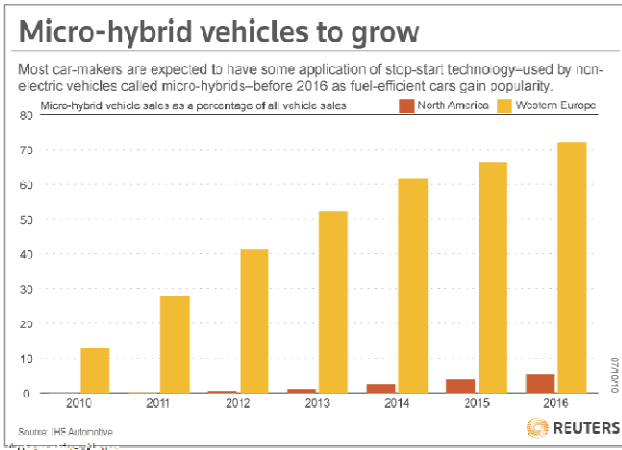


Figure 1. Projections of micro-hybrid vehicles in North America and Europe (reprinted in [7])

2 Vehicle Modeling

This section outlines the key multi-domain component and subsystem models that support the subsequent vehicle drivability applications. The main model components are detailed as are the different modeling approaches that can be used to support vehicle drivability applications.

2.1 Vehicle Model Architecture

The ability to create configurable model architectures in Modelica is one of the key enablers for architecture-driven development in model-based systems engineering [6]. With core language support for model management and configuration and formal interface definitions, Modelica provides an excellent foundation for distributed, collaborative systems modeling.

The Vehicle Dynamics Library takes an architecture-driven approach to model development and configuration. One of the fundamental guiding principles of VDL is the ability to mix behavioral and physical models to conveniently change between different configurations and also between different levels of detail. From a powertrain perspective, the use of the Rotational3D library [9] is key as it provides a fully-defined representation in 3D that can easily be reduced to a 1D representation or vice versa. Within the same architecture, the VDL can support 1D modeling typically used for conceptual representations early in the product development process to represent the main degrees of freedom to gain early understanding and understand system-level interactions. Furthermore, full 3D representations can be supported which require more extensive parameterization but with a level of detail that provides virtual testing capabilities.

VDL makes use of these Rotational3D connectors to represent the interface cuts between the different components in the powertrain, from the engine to the wheels. Therefore, the architecture has inherent support for plug-and-play compatible exchange between 1D and 3D components. For example, it is straightforward to combine a 1D driveline in a 3D chassis or a detailed engine model on a lumped chassis.

The component-based interfaces also make it straightforward to switch context, as illustrated for the engine and transmission in Figure 2. In the top model, the engine and transmission are used in a full vehicle template, connected via the driveline to the wheels and the chassis. The other model contains only the engine and the transmission connected to a load, which then can be very simplified, e.g. just a 1D mass, or also a full chassis with driveline.

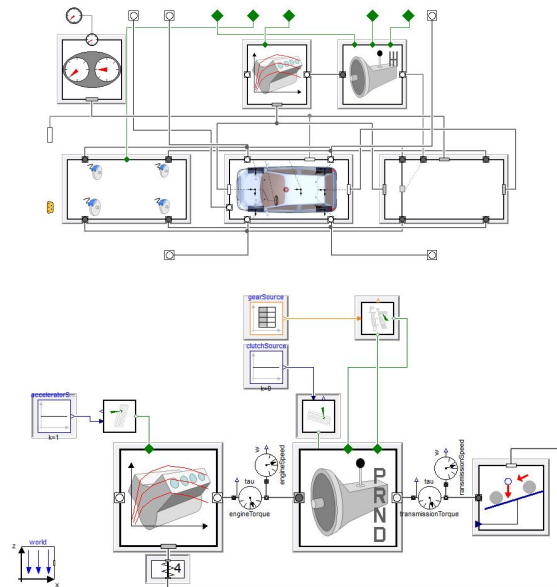


Figure 2. Architectures illustrating the same engine and transmission models in different contexts as shown by the models in a full vehicle representation (top) and together with a load representing the vehicle (bottom)

2.2 Engine

Modelica has been used extensively for simulating detailed engine transient response including combustion [2] [3]. Proper representation of the engine dynamic response is critical for vehicle drivability efforts. Both the mean and fluctuating component of the engine torque production can induce undesirable vehicle driveline response. The various delays in the engine due to controls scheduling constraints, actuator response, air path dynamics, and fueling dynamics can also be important. Engine inertial response (piston mass, crank/slider inertia, crankshaft inertia) can also be an important consideration in dynamic

simulation, especially for cranking, launch, and start-stop events.

Typical engine modeling approaches for drivability applications include the following:

- Mean value modeling based on maps and actuator inputs
- Mean value modeling including air path (intake and exhaust) dynamics
- Mean value modeling with superimposed torque fluctuations at the crankshaft
- High frequency modeling with multiple cylinders based on input cylinder pressure
- High frequency engine modeling with multiple cylinders and physics-based combustion modeling

These approaches cover a range of predictive capability, computational effort, and input/calibration data requirements. Furthermore, the level of expertise required to implement and validate the various models varies greatly.

The engine components in the Vehicle Dynamics Library support multiple approaches for modeling the engine and crankshaft. These approaches range from a pure map-based mean value approach to a high frequency approach based on cylinder pressures. Figure 3 shows an I4 engine with a cylinder-based pressure calculation and a distributed model of the engine bottom. An analytic representation can be used for the pressure calculation to allow faster simulation with a standard table for torque as a function of throttle and engine speed required for parameterization. This model calculates a pressure trace profile using spark timing with adjustable shape parameters and induces the appropriate amount of crankshaft torque fluctuation. A sample cylinder force and engine animation for the I4 engine is shown in Figure 4. A full tabular representation for the cylinder pressure as a function of crankangle, throttle, engine control settings, and engine speed can also be used.

To support additional modeling options for the engine, engine models from the Engine Dynamics Library [8] can be integrated into the VDL vehicle architecture. The Engine Dynamics Library currently provides mean value engine modeling capability including the air path dynamic effects, exhaust modeling, and thermal effects. The focus of the library is real-time like simulation of gas exchange and mean value torque production to support engine optimization and evaluation of engine control strategies. A turbocharged, spark-ignited engine model using EDL is detailed in Section 3 and integrated within the VDL architecture for use in a start-stop application.

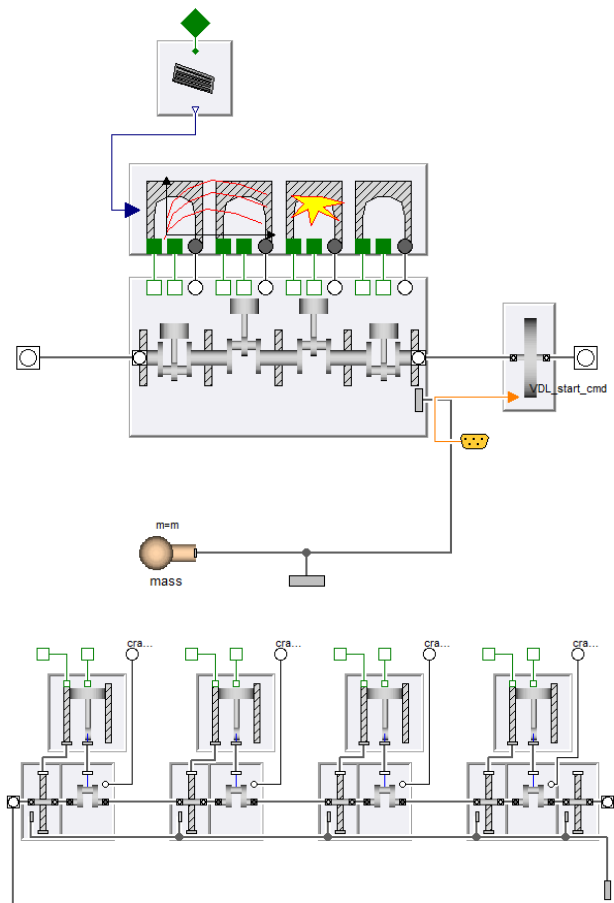


Figure 3. I4 engine with cylinder pressure calculation and dynamic bottom (top) and detail for engine block showing piston and crankshaft models (bottom)

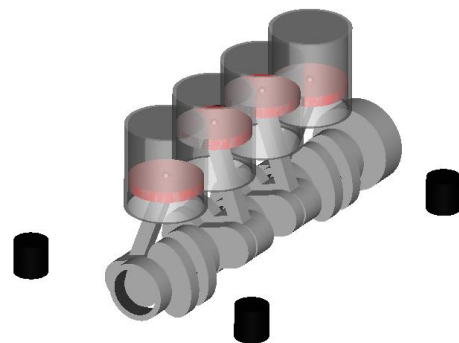
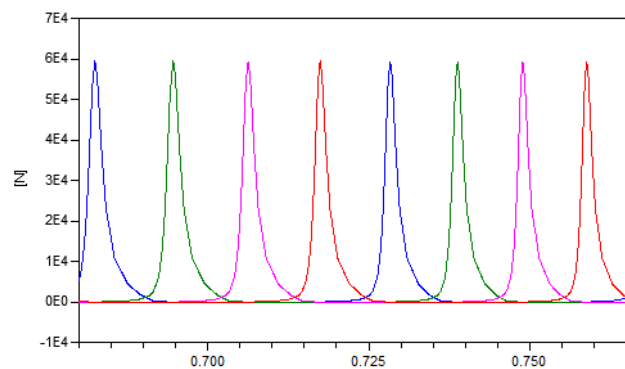


Figure 4. Cylinder force (top) and engine animation (bottom) for I4 engine

2.3 Transmission and Driveline

Several transmission implementations are available in the Vehicle Dynamics Library, including representations of automatic, manual, and CVT transmissions. Within the architecture of Vehicle Dynamics Library, it is certainly possible to implement custom transmission models including both 1D and 3D effects, backlash, friction, etc.

Dual clutch transmissions (DCT) are becoming increasingly popular due to the projected fuel economy benefits resulting from the removal of the torque converter, dry clutch technology, etc. However, many of these changes also can pose new performance (no torque multiplication from torque converter), drivability (no damping from torque converter due to fluid coupling), and control (managing clutch to clutch transitions for gear changes) challenges. Thus, dual clutch transmissions are often mated with a dual mass flywheel to provide the required damping but with increased inertia.

To illustrate some of these challenges in the following drivability applications, the simple dual clutch gearbox model shown in Figure 5 was implemented and integrated into a transmission model. Since this model is primarily for demonstration purposes, the control interfaces are simplified. This model implementation should also be considered as functional as no detailed parameterization data (clutch, gearing, etc.) was available to support a more detailed model implementation for the purposes of this paper. For a full treatment of a dual clutch transmission and associated control, the interested reader is referred to [10].

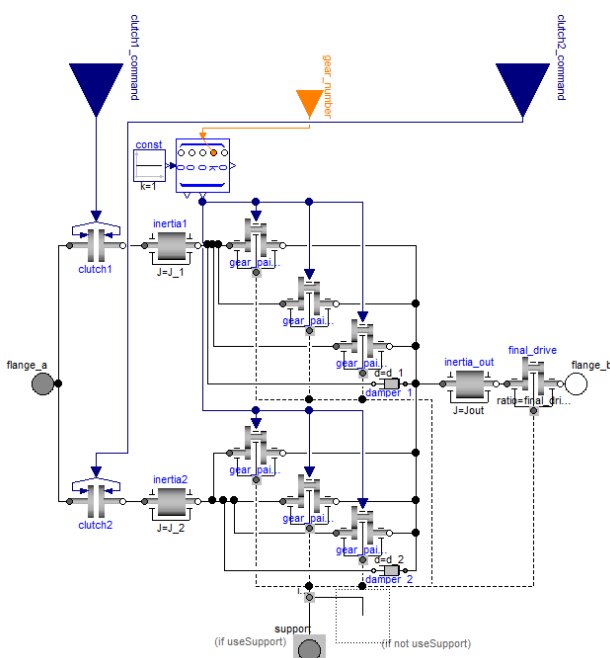


Figure 5. Dual clutch transmission gearbox

The Vehicle Dynamics Library provides both components and assembled subsystems to model various driveline implementations (front, rear, all-wheel drive). Components are available for gears, gear pairs, clutches, shafts, differentials, etc. The shaft models are implemented such that geometric effects such as joint effects and bend angles can be modeled if needed. Pure 1D rotational components can be used as well with the 1D/3D structure provided by the library architecture. Figure 6 shows a rear wheel drive driveline model with a transmission shaft, rear differential, and geometric half shafts; this model is used in the vehicle model examples shown in Section 3.

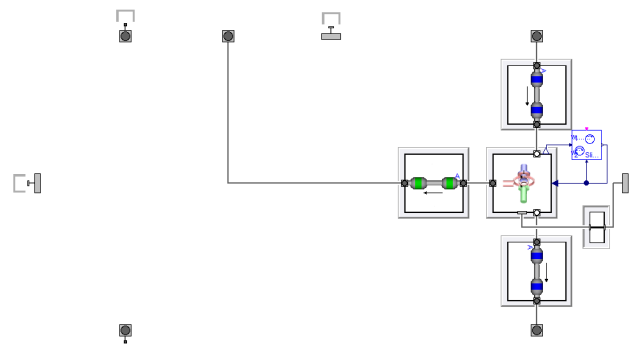


Figure 6. Rear wheel drive driveline

2.4 Chassis

The Vehicle Dynamics Library includes a wide range of suspension models with fidelity levels that span from planar models to fully geometric, elastokinematic models. Lower fidelity suspension models are more desirable in drivability simulations for many reasons. First, the engineers performing drivability simulations are mainly interested in straight-line behavior and longitudinal dynamics. For these types of simulations, unlike in handling simulations, it is not important to accurately represent how the wheel moves with respect to the chassis for a given wheel travel and load condition. A key benefit of using lower fidelity suspension models is that information required to represent them is significantly lower than a more complex physical model.

Historically, lower fidelity models have included the following representations:

- Planar suspension
- Equivalent roll stiffness
- Lumped mass
- Swing arm

Each of these suspension models are available to be used within VDL. The lower fidelity suspension models have the same interface as the more complex physical suspension models. This modular approach

ensures that either representation can be used in the chassis at any time.

The range of chassis implementations are described in more detail below and shown in Figure 7 via animations from VDL. The example applications in Section 3 include simulations within this range of chassis models.

Pitch model: The suspension is modeled as a rigid axle that can only translate vertically with respect to the chassis body. Ride stiffness is modeled using vertical springs and dampers. This model allows the chassis to heave and pitch.

Equivalent roll stiffness model: The suspension is modeled as a rigid axle that rotates about a single axis. Roll stiffness is modeled using a torsional spring and damper. This model allows the chassis to roll.

Lumped mass model: The suspension is modeled to allow each wheel to translate vertically with respect to the chassis body. Ride and roll stiffness is modeled using vertical springs and dampers. This model allows the chassis to heave, pitch and roll.

Swing Arm model: The suspension is modeled to allow each wheel to swing on a control arm about a single axis. Ride and roll stiffness is modeled using vertical springs and dampers. This model allows the chassis to heave, pitch and roll.

Tabular model: The suspension is modeled using tables that define the kinematic and compliant model of the wheel with respect to the chassis body. The model allows the chassis to heave, pitch and roll.

In order to use these models to represent a chassis for drivability work, it is only necessary to provide the chassis mass, track width, wheelbase and approximate spring rates. Since many engineers who work on powertrain response and drivability applications do not have ready access to the geometry required for more detailed chassis models, a small set of parameterization data can result in significant reductions in model development time.

While lower fidelity dynamic models typically are sufficient for vehicle drivability work, there are some applications which might require more detailed chassis representations. For example, some launch maneuvers might require more detailed models to observe anti-squat or differential/axle windup effects. With the architecture and associated components from the Vehicle Dynamics Library, the various chassis representations are plug-in compatible such that full multibody representation can be seamlessly integrated.

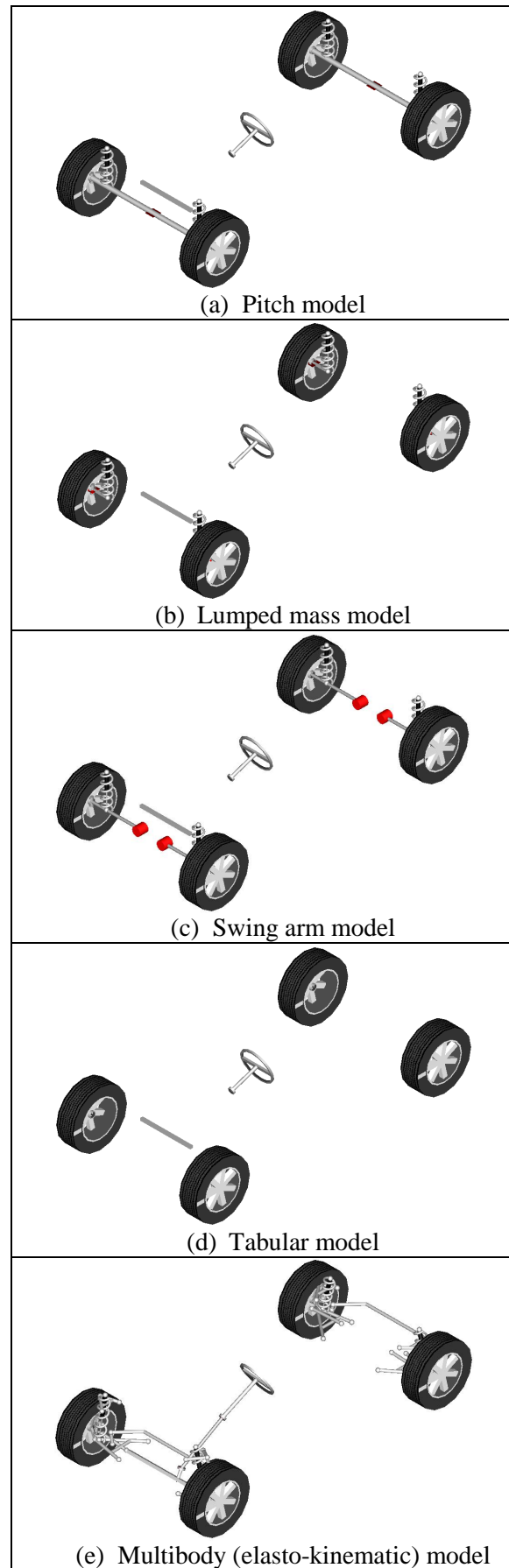


Figure 7. Range of chassis implementations from VDL

2.5 Powertrain Mounts

Powertrain mounts are another critical element for modeling of vehicle drivability events. The movement of the powertrain on the mounts affects the torque transfer in the driveline as well as providing a transfer path for vibrations to the vehicle seat track. Mount design affects vehicle performance and comfort and requires a simulation environment capable of transient simulations over critical maneuvers of interest (idle, launch, tip in- tip out, etc.).

The VDL architecture includes a configurable component for modeling the powertrain mounts. The inherent 3D support makes addition of reaction forces and torques straightforward to capture the true dynamics of the system. The component can be configured for the number of mounts, mount locations, and also the characteristics of the mount behavior. For example, Figure 8 shows bushing compression (gray areas) due to rotation of the differential housing when the torque is transmitted from the longitudinal to lateral direction. These effects require a 3D representation of the driveline and differential which are readily implemented using components from the Vehicle Dynamics Library.

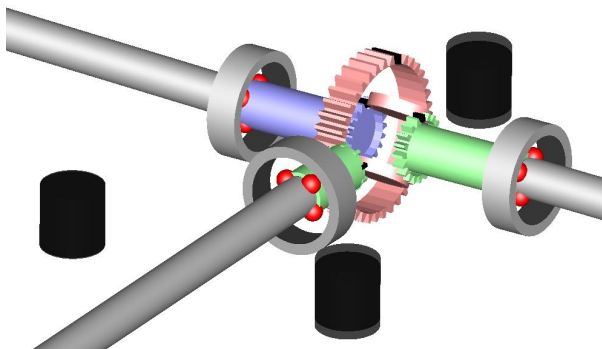


Figure 8. Differential housing wind-up

3 Application Examples

Using the component models outlined in the previous section, this section provides several example models illustrating the impact of modeling approaches on vehicle drivability response.

3.1 Vehicle Launch

Figure 9 shows a model configured for wide open throttle (WOT) vehicle launch from idle with an automatic transmission. This implementation includes a simple mapped engine. Figure 10 shows some sample results from the launch and subsequent shift events with the tabular chassis model. Note the acceleration disturbances around each shift event. The-

se sorts of disturbances can be mitigated by appropriate torque control and shift coordination between the engine and transmission. The development and optimization of such coordinated control is readily achieved using model-based systems engineering approaches with VDL.

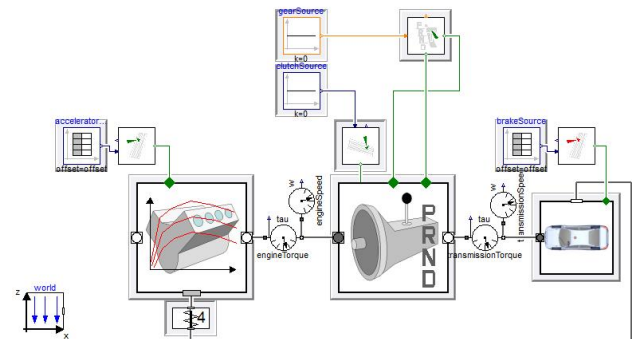


Figure 9. Vehicle launch model with automatic transmission

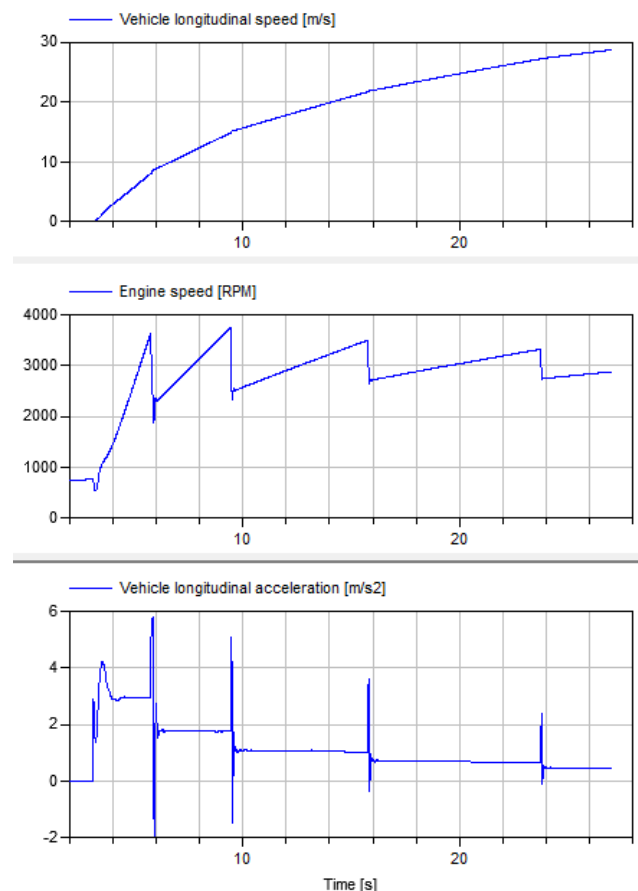


Figure 10. Vehicle launch from idle and shift results with automatic transmission

Figure 11 shows comparisons between the chassis pitch and roll angle for the various chassis model implementations. For the chassis pitch angle, the pitch, lumped mass, and swing arm models provide similar results as do the tabular and multibody mod-

els. For the chassis roll angle, the pitch model shows essentially no roll while the results from the lumped mass and swing arm models are grouped together as are the results from the tabular and multibody models.

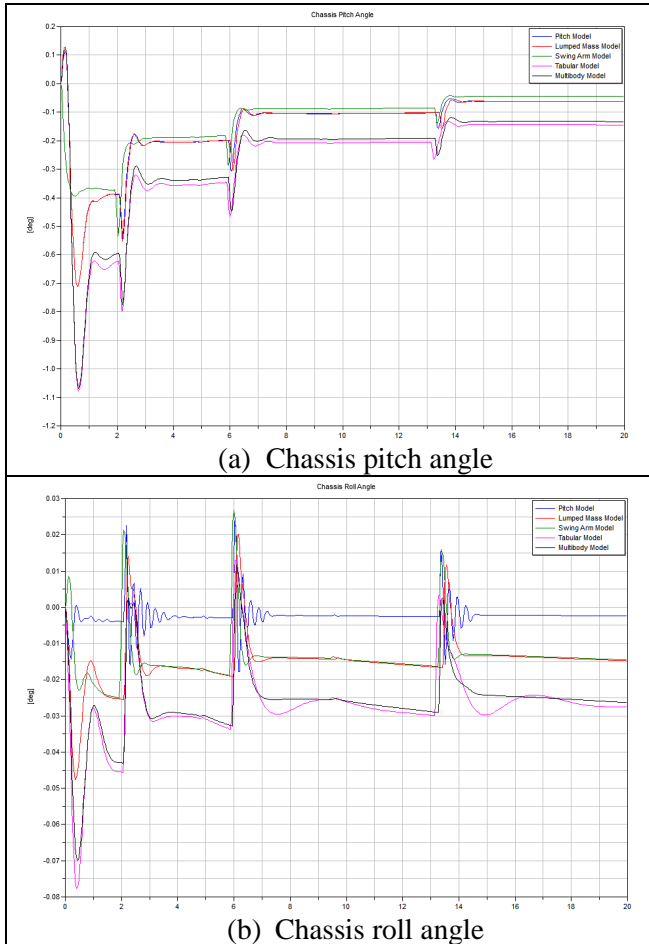


Figure 11. Chassis pitch (a) and roll angle (b) for the various chassis model implementations

Figure 12 shows a model configured for wide open throttle vehicle launch with a dual clutch transmission. Note that the only changes from Figure 9 are the transmission swap and associated transmission control specification (open loop in this example). Figure 13 shows some sample results from the initial launch and first 1-2 shift event. Note the driveline disturbances shown during the shift due to a poorly executed clutch to clutch transition. With a dual clutch transmission, the key to shift feel and performance is managing this transition thus highlighting the importance of clutch modeling (frictional characteristics, actuation, dynamic response, etc.) and controls for this type of transmission. Given the complex dynamic response, a model-based systems engineering approach is required for multi-attribute balancing of performance and drivability.

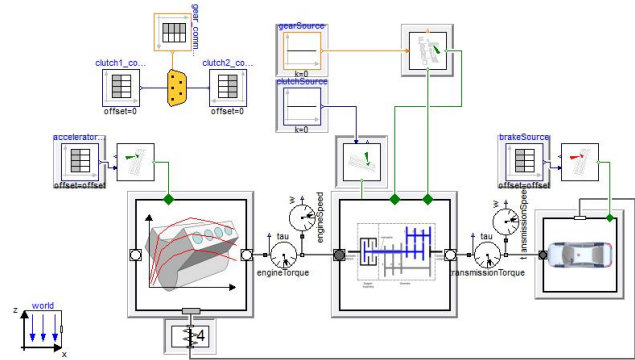


Figure 12. Vehicle launch model with dual clutch transmission

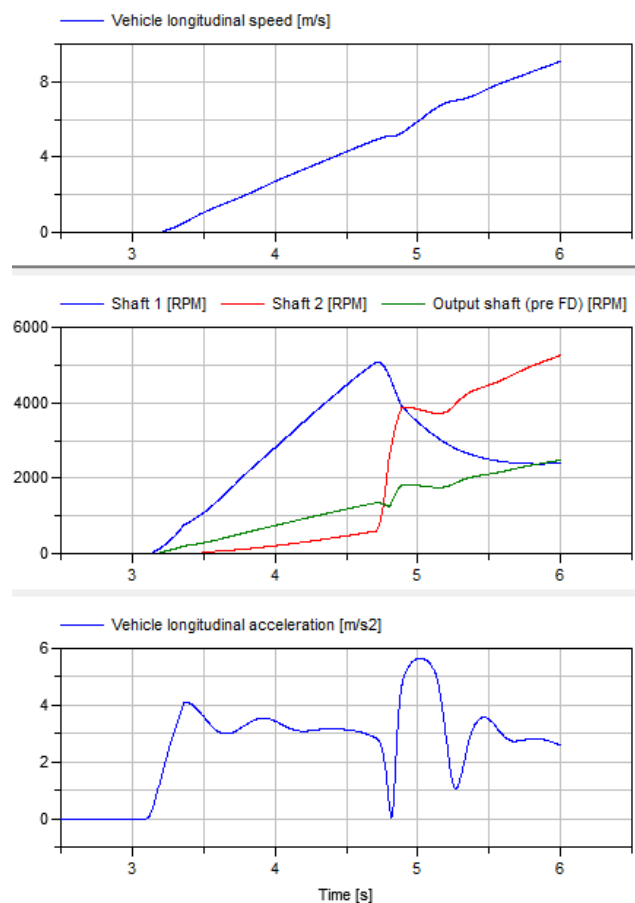


Figure 13. Initial launch and 1-2 shift results for dual clutch transmission

The vehicle launch models illustrate the power of a modeling architecture and supporting component models such that the focus of the model can easily be shifted from full vehicle to powertrain with different levels of complexity for the chassis and driveline. This approach allows model detail to be implemented in the areas where it is critical for observing the dynamic phenomena of interest while allowing model simplifications in other areas. This approach enables a balance between model complexity, computa-

tional effort, and also parameterization effort given that more detailed models typically require more detailed data for parameterization.

3.2 Start-Stop

Start-stop technology on mild/micro-hybrid vehicles offers compelling fuel economy benefits with elimination of idle fuel consumption. While the fuel economy benefits are clear, the drivability impact can be significant if the restarts are not managed well. Customers experience many more starting events, and their expectations regarding these events differ when driving as compared with a single start in park in a garage or parking lot. The engine must be quickly cranked from rest and able to meet driver demand for the subsequent launch event. The engine cranking, fueling, airflow, and transmission engagement events must be managed to provide quick restart performance while minimizing driver disturbances felt at the seat track.

As with all drivability applications, the appropriate choice of model for the engine, transmission, driveline, and chassis depends on the overall goal of the simulation and frequency range of interest. Potential applications include the following:

- Starter motor, battery, and electrical system sizing
- Model-based controls development
- Design of driveline isolation components (damper, dual mass flywheel, etc.)
- Powertrain mount design
- Launch performance and sensitivity to physical and controls parameters
- Driveline response over the range of engine speeds and torques seen during crank, initial combustion, and run-up phases

For start-stop applications, one of the key modeling choices involves the engine dynamic response and resulting torque signature. While mean value modeling approaches may be sufficient for some applications, others may require that the torque pulses at the crankshaft are represented as they may be key for the drivability phenomena of interest. The start-stop examples that follow cover a range of engine modeling approaches from the Vehicle Dynamics and Engine Dynamics Libraries.

As described in Section 2.2, VDL includes engine models capable of producing fluctuating torque at the crankshaft. Using the model shown in Figure 3 with a separate cylinder head and distributed engine bottom, a map-based mean torque can be analytically transformed into a pressure calculation including the influence of control parameters on the pressure shape. This representation, while approximate when

compared to a detailed engine cycle simulation, does not require detailed engine characterization for model development (i.e. intake and exhaust system flow characteristics, valve profiles and discharge coefficients, combustion characteristics, etc.) and can provide the appropriate amount of crankshaft torque fluctuation in a computationally efficient manner. It should be noted that appropriate care must be taken when calculating the mean torque to account for the various delays in the engine response which are not physically modeled (i.e. throttle and airflow response, fueling response, manifold filling and emptying, etc.).

Within the architecture provided by VDL, a plug-compatible I4 engine model with fluctuating torque is chosen for the replaceable engine subsystem shown in Figure 9. Results from start-stop simulations with this engine with an automatic transmission are shown in Figure 14. These results show the impact of the starter on initial launch behavior. When the starter disengages early, the engine speed drops after the initial crank until the runup phase begins due to the initial firing events. With normal starter disengagement, the engine speed smoothly increases during crank followed by the initial combustion events. If the starter torque is also increased, the expected increase in engine speed is observed. Note the similar trends in the vehicle speed response.

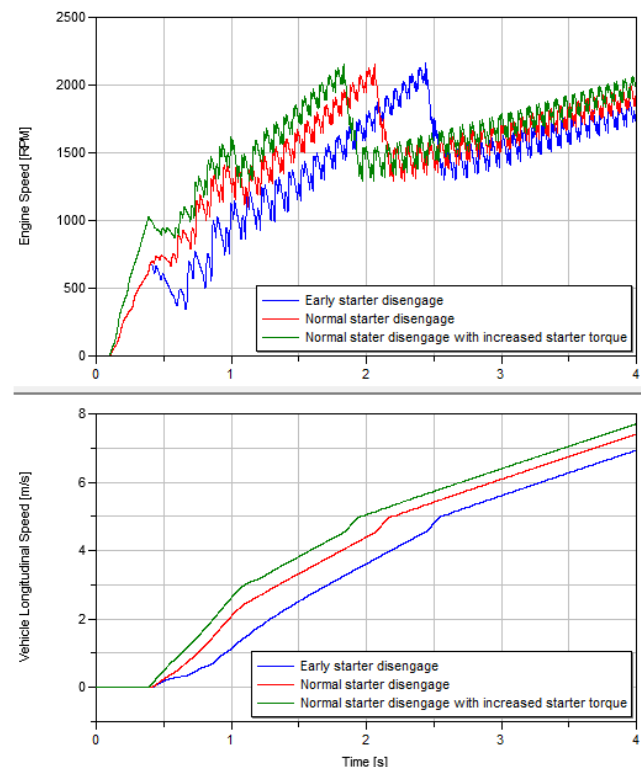


Figure 14. Start-stop response with automatic transmission [engine speed (top) and vehicle speed (bottom)]

To support additional modeling options for the engine, engine models from the Engine Dynamics Library [8] can be integrated into the VDL vehicle architecture. The Engine Dynamics Library currently provides mean value engine modeling capability including the air path dynamic effects, exhaust modeling, and thermal effects. With the physical modeling approach in the EDL, the engine model responds naturally to changes in actuation and control commands based on the individual component characteristics. Appropriate delays in engine response are also simulated via the physical characteristics of the components. The component-based approach in EDL also allows advanced concept evaluation, component sizing and optimization studies, and model-based controls development.

Figure 15 shows a turbocharged, spark-ignited engine model using EDL. The model includes lumped representations of the intake air path (light blue connections), exhaust air path with EGR loop (orange connections), simplified cooling path (blue connections), key heat transfer effects in the charge air cooler, cylinder, and exhaust manifold, and a turbocharger with wastegate. The model is roughly parameterized for a 2L engine. For the purposes of this paper, a battery and electric motor are added to the turbo system to provide “eboost” capability. This engine model is then integrated into the VDL architecture with the dual clutch transmission shown in Figure 12.

Figure 16 shows results from the start-stop launch with the EDL engine model, dual clutch transmission, and tabular chassis model. The results are for an aggressive launch with varying levels of eboost motor assist applied in the first second of the launch event. The simulations show the initial cranking event, engine run-up due to combustion, and 1-2 shift with the dual clutch transmission which occurs around 1.5s. With increasing motor assist, the turbo speed increases much more rapidly than would be possible with the compressor driven by the turbine alone, especially since the early combustion events do not provide significant exhaust enthalpy to drive the turbine (i.e. lower mass flow rates and lower exhaust temperatures). The increased turbo speed results in additional boost and thus additional engine torque as shown in Figure 17 and higher engine speeds.

With a dual clutch transmission, the coupling between engine and transmission can be managed to optimize the overall launch event by controlling the timing of wheel torque subject to drivability constraints.

While the EDL currently provides a mean value modeling approach for the engine, it is also possible to generate a fluctuating torque using the same mod-

els shown in Figure 3 by replacing the tabular torque map from VDL with the dynamic EDL model. With this approach, the engine model remains physics-based but can also provide fluctuating crankshaft torque for drivability applications without significant additional computational expense.

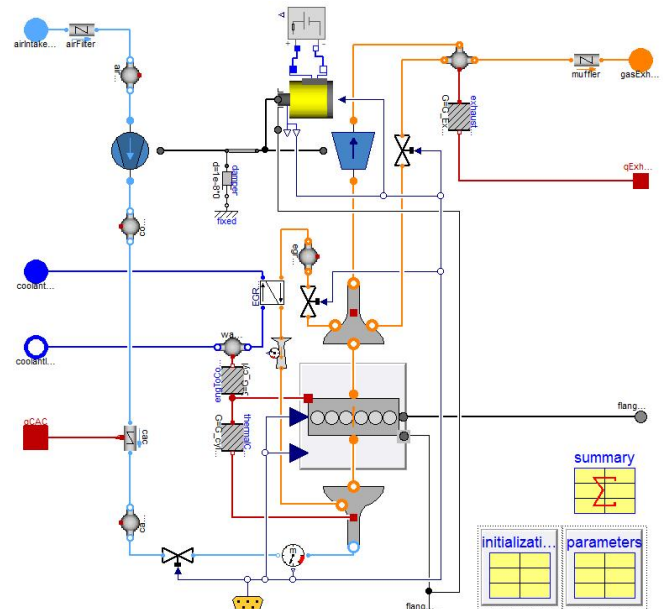


Figure 15. Turbocharged, spark-ignited engine model using the Engine Dynamics Library

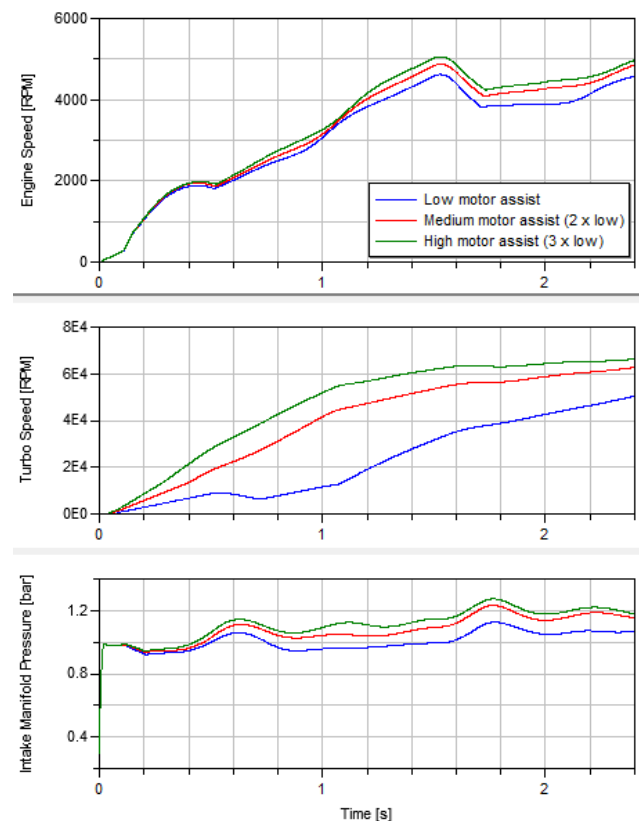


Figure 16. Start-stop response with Engine Dynamics Library engine model and dual clutch transmission

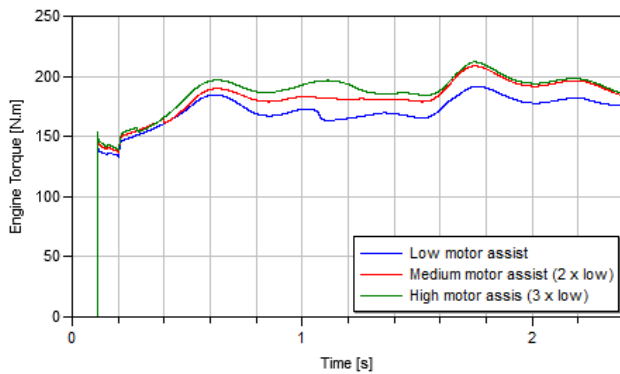


Figure 17. Engine torque for start-stop launch

4 Conclusions

Several application examples focused on vehicle drivability have been detailed. These application examples include vehicle launch, start-stop, and transmission shift performance. These examples highlight the use of sophisticated model libraries with different levels of fidelity for key components such as the chassis along with different modeling approaches for the engine and both automatic and dual clutch transmissions. The model libraries illustrate the multi-domain approach required to simulate vehicle drivability. Using the flexible model architecture from the Vehicle Dynamics Library, the various examples are seamlessly configured using plug-compatible variants. The examples are structured to illustrate how increasingly sophisticated models provide additional model fidelity or increase the drivability phenomena observed. An engine model created using the Engine Dynamics Library is coupled with the models and architecture from the Vehicle Dynamics Library to illustrate a range of engine modeling approaches to support vehicle drivability applications.

References

- [1] Andreasson, J., "The Vehicle Dynamics Library: New Concepts and New Fields of Application", *Proceedings of 8th International Modelica Conference*, 2011.
- [2] Newman, C., Batteh, J., and Tiller, M., "Spark-Ignited-Engine Cycle Simulation in Modelica", *Proceedings of 2nd International Modelica Conference*, pp. 133-142, 2002.
- [3] Batteh, J. and Newman, C., "Detailed Simulation of Turbocharged Engines in Modelica", *Proceedings of 6th International Modelica Conference*, pp. 69-75, 2008.
- [4] Wang, et al., "Integrated Thermal Management Simulations: Evaluating the Effect of Underhood Recirculating Airflows on AC-System Performance", *Proceedings of 7th International Modelica Conference*, pp. 413-422, 2009.
- [5] Eborn, et al., "AirConditioning - a Modelica Library for Dynamic Simulation of AC Systems", *Proceedings of 4th International Modelica Conference*, pp. 185-192, 2005.
- [6] Batteh, J. and Tiller, M., "Implementation of an Extended Vehicle Model Architecture in Modelica for Hybrid Vehicle Modeling: Development and Applications", *Proceedings of 7th International Modelica Conference*, pp. 823-832, 2009.
- [7] Niedermeyer, E., "The Shocking Truth About Start-Stop Systems", <http://www.thetruthaboutcars.com/2011/06/the-shocking-truth-about-start-stop-systems/>, June 29, 2011.
- [8] Andersson, D., and Dahl, J., 2012, "Gas Exchange and Exhaust Condition Modeling of a Diesel Engine using the Engine Dynamics Library", *Proceedings of 9th International Modelica Conference*, 2012.
- [9] Andreasson, J. and Gäfvert, M., "Rotational3D-Efficient modelling of 3D effects in rotational mechanics", *Proceedings of 6th International Modelica Conference*, pp. 515-520, 2008.
- [10] Isernhagen, H. and Guhmann, C., "Modelling of a Double Clutch Transmission with an Appropriate Controller for the Simulation of Shifting Processes", *Proceedings of 6th International Modelica Conference*, pp. 333-339, 2008.

Status of ClaRaCCS: Modelling and Simulation of Coal-Fired Power Plants with CO₂ Capture

Johannes Brunnemann^{*1}, Friedrich Gottelt¹, Kai Wellner², Ala Renz¹, André Thüring⁴, Volker Roeder³, Christoph Hasenbein³, Christian Schulze⁴, Gerhard Schmitz² and Jörg Eiden¹

¹XRG Simulation GmbH, Harburger Schlossstraße 6-12, 21079 Hamburg, Germany

²University Hamburg-Harburg, Inst. of Thermo-Fluid Dynamics, Denickestr. 17, 21073 Hamburg, Germany

³University Hamburg-Harburg, Inst. of Energy Systems, Denickestr. 15, 21073 Hamburg, Germany

⁴TLK-Thermo GmbH, Hans-Sommer-Str. 5, 38106 Braunschweig, Germany

Abstract

Within the DYNCAP project, the Modelica library ClaRaCCS is being developed. This library will provide a framework to model both steam power plants and carbon capture units in an integrated manner. The current status of the library is presented. The structure of the library and the general model design is outlined. Its user-friendly handling as well as its high flexibility in the modelling of individual complex scenarios are demonstrated by the concrete modelling of a furnace. The scenario of a closed steam cycle coupled to a carbon capture cycle based on an amine gas treatment is described and simulation results are briefly discussed.

Keywords: power plant; Clausius-Rankine cycle; CO₂ capture; CCS; amine gas treatment; transient simulation; library

1 Introduction

The ongoing climate change is a serious ecological and economical challenge in the next decades. The Intergovernmental Panel on Climate Change (IPCC) recommends a reduction of CO₂ emissions to below 80% until 2050 compared to 1990 [1]. Although the proportion of renewable energies is growing significantly, fossil fuels such as coal will remain central to the world's energy supply during the next decades.

It is therefore necessary to evaluate power plant technologies appropriate for a significant reduction of CO₂ emissions in the short term. One already available technological solution is the capture of CO₂ from flue gases of fossil-fuelled power plants and its storage (CCS). This technology has to be embedded into

a future energy mix with a large percentage of renewable and fluctuating energies, such as wind and solar power. Hence, the need for a flexible operation of conventional fossil-fuelled power plants under rapid, large and frequent load changes arises.

The evaluation of such variable operation scenarios requires a thorough investigation of future power plant dynamics. This shall result in recommendations for the design and operation of power plants, that meet certain objectives regarding efficiency, technical limitations and ecological standards. A valuable tool to tackle this challenging task is computer simulation.

As a part of COORETEC [2], an initiative of the German Federal Ministry of Economics and Technology, the project DYNCAP [3] aims at studying the dynamic behaviour of steam-power processes with CO₂ capture in order to provide balancing energy. The project started in March 2011 and will be finished in September 2014. One major outcome of the project is the Modelica library ClaRaCCS (Clausius-Rankine with CO₂ Capture and Storage). The goal of the library is to provide models for the analysis of complex power plants with CO₂ capture in both static and dynamic operation mode. After completion of the DYNCAP project, the library will be freely available under the Modelica license. The current development is performed using Dymola [4], however the final version of the library is intended to work with SimulationX [5] as well.

This paper gives an introduction to ClaRaCCS and presents the current status of development. The paper is organised as follows: Section 2 summarises the technical fundamentals for conventional steam power plants as well as carbon capture processes modelled in the library. In Section 3 general properties of the li-

*brunnemann@xrg-simulation.de

brary are introduced: Starting from the general library structure the guiding principles, that underlie the models in ClaRaCCS, are explained. The treatment of media data as well as validation of models will also be outlined. The described properties will then be illustrated by a concrete modelling example in section 4, where the model of a furnace is described. Section 5 demonstrates the current status of ClaRaCCS by giving an example of use: the model of a coal-fired power plant with attached carbon capture unit is presented. The results of a simulation scenario are shown, where throttling of the carbon capture unit is used in order to meet the demand for a short term increase of the generator power output of the plant. Finally, section 6 gives a summary and outlines future steps of development.

2 Technical Background

The processes covered in ClaRaCCS are conventional hard-coal-fired power plants and their derivatives for CO₂ capture, the Post-Combustion Capture process (PCC) and the Oxyfuel process (further information in [6]). Lignite-fired power plants and gas-fired combined cycle power plants are not part of the project DYNCAP, but may be included in the future.

Because the model implementation of the Oxyfuel process is still work in progress the respective section will only give a short overview. The section of the PCC process will introduce a little more of this technology because it is part of the simulation example given in section 5.

2.1 Conventional Hard-Coal-Fired Power Plants

State-of-the-art conventional hard-coal-fired power plants burn pulverised coal in the steam generator with air. The heat is transferred to a steam cycle that converts it to electric energy. The general simplified process scheme of the power plant is shown in figure 1. Mills pulverise and dry the raw hard-coal. The raw flue gas contains, additionally to nitrogen, CO₂, oxygen and water, also certain amounts of nitrogen oxides, fly ash (dust) and sulphur oxides. Therefore the flue gas treatment comprises a denitrification system, an electrostatic precipitator and a wet desulphurisation unit.

The steam cycle comprises a super-heater and a re-heater. Power plants currently under construction have a live steam (high pressure) pressure of about 285 bar and a live steam temperature of about 600 °C. The

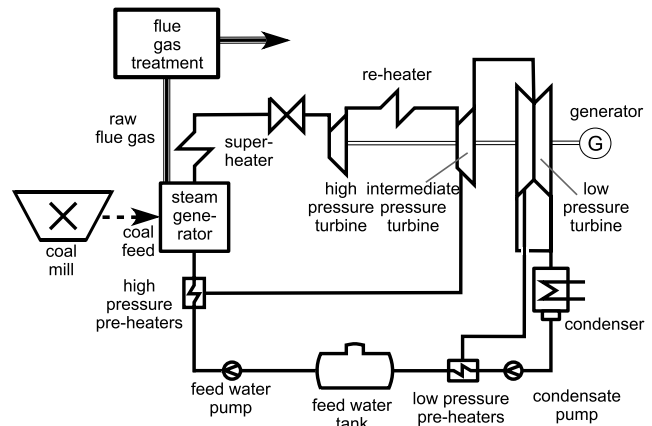


Figure 1: Simplified process scheme of a conventional power plant.

re-heated steam is in the range of 60 bar and 620 °C. Feed water is pre-heated with steam in the low pressure range and the high pressure range. Up to nine feed water pre-heaters are implemented, each with a steam tapping from the turbines or crossover sections. The turbines are all coupled with the same shaft and running at constant speed in normal operation as they are directly coupled with the electric net by the generator.

The overall efficiency of such a power plant is approx. 46 %.

2.2 Post-Combustion Capture Process

In a PCC CO₂ is separated from the flue gas of a conventional coal-fired steam power plant by a chemical absorption-desorption process. The reduction of the CO₂ emissions is accompanied by a significant loss in the electrical power output and a related net efficiency penalty of 8-12 %-pts. As reference the solvent Monoethanolamine (MEA) is used with a solvent mass fraction of 30 % MEA and a targeted CO₂ capture rate of 90 %.

Figure 2 shows the schematic PCC process. The flue gas of the boiler passes through the flue gas treatment, where it is cleaned, and then enters the absorber column at the bottom, in which the CO₂ is absorbed by a counter current solution flow. The treated gas is released to the atmosphere, while the rich (CO₂-loaded) solution leaves the absorber at the bottom. Downstream the absorber, the rich solution is pumped through the rich-lean heat exchanger, heated up and enters the desorber column at the top. In the desorber the absorbed CO₂ is stripped from the rich solution. The required heat duty is provided by a reboiler in which steam from the power plant is condensed. From

the bottom of the desorber, the lean solution is pumped to the entrance of the absorber, passing the rich-lean heat exchanger where it is cooled and pre-heats the rich solution. The captured CO_2 , nearly pure, is compressed and pumped to the storage. A detailed explanation of the process can be found in [7]. An overview of this process is given in [8].

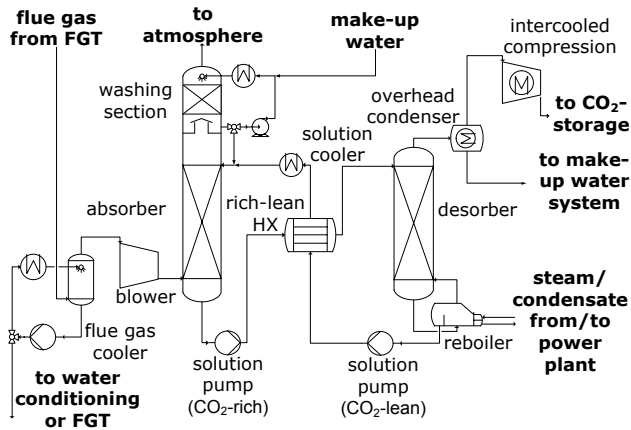


Figure 2: Flow sheet of the PCC process, cf. [7].

The main task of the PCC is the reduction of the CO_2 -emissions by a certain value. The CO_2 capture rate depends on the circulated solution flow rate and the working capacity of the solution. Here the working capacity is defined as the difference between CO_2 loadings behind the absorber and behind the desorber. The solution rate can be influenced by the pump upstream the absorber. The lean loading depends on the reboiler heat duty and thus is directly affected by the steam mass flow which is condensed in the reboiler. The liquid level of the absorber sump is controlled by a pump that conveys the solution to the desorber.

2.3 Oxyfuel Process

Conventional coal-fired power plants and power plants with PCC burn the coal with air. On the contrary, in the Oxyfuel process the coal is burnt in an atmosphere of oxygen from an air separation unit mixed with recirculated flue gas. As the nitrogen of the air is avoided in the combustion process, the flue gas contains mainly CO_2 (70 vol.-%), water and small amounts of oxygen, nitrogen and argon. The flue gas fraction that is not recirculated is treated to remove the impurities in order to receive a CO_2 stream with a purity higher than 96 %. The overall power plant net efficiency is decreased by approx. 8-10%-points when the Oxyfuel process is applied. This includes the cryogenic air separation unit and the compression of the captured CO_2 to a pressure

of 110 bar. A detailed overview of the Oxyfuel process can be found in [9].

3 The ClaRaCCS Library

3.1 Library Structure

Creating a library covering a very broad range of physics that is at the same time well-arranged and user-friendly, demands an elaborate library structure. Figure 3 shows the top level content of the ClaRaCCS library. Beside the usual packages like *UsersGuide*, *Examples* and *Media* the library is structured into the main existing functional groups of the physical processes under consideration. *Components* is the package with the most basic models describing e.g. turbo machines, furnace, heat exchangers or thermal separation. In *SubSystems* these components are used to create more complex models e.g. a boiler or an air separation unit. The package *SubProcesses* then in turn contains models which are built from *SubSystems* models like whole CO_2 capture cycles. *PowerPlants* consists of models representing whole power plants and is the package with the most complex models.

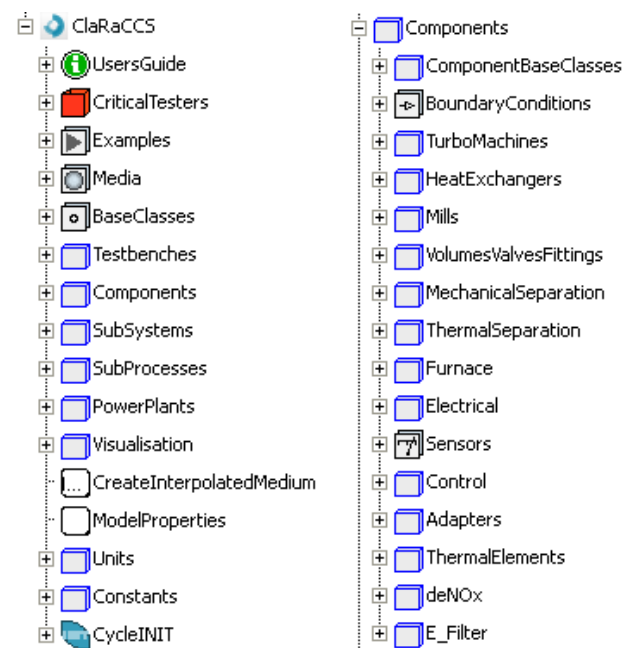


Figure 3: Top level content of the ClaRaCCS library and the central component package.

3.2 Model Design Principles

When setting up the model of a complex physical system such as a power plant, the first question to be answered is what physical fidelity is needed to cope with

the given simulation task. The answer to this question refers to the **level of detail** necessary for each component and sub-process. The next step is to define the general **physical effects to be considered** for solving the given task. Finally, the **level of physical insight** into the considered physical aspects must be chosen.

In what follows it will be explained how these three stages guide the model design of the ClaRaCCS library. For illustration the concept will be applied to the well-known example of a fluid flow in a pipe.

3.2.1 Level of Detail

In [10] a classification of component models into different levels of detail was developed. It is mainly based on two criteria:

- Purpose of model. In which simulation context will the model be used? What questions and physical effects shall be analysed with the model?
- Applicability of model. What are the main assumptions the model is based on? Are there some structural limitations?

The model design of ClaRaCCS has been inspired by these ideas. Moreover it aims to provide a well balanced combination of readability¹, modelling flexibility and avoidance of code duplication. Consequently, each component in the ClaRaCCS library is represented by a family of *freely exchangeable* models. Every component family is grouped into **four levels of detail**:

L1. Models are based on characteristic lines and / or transfer functions. This results in an idealised model, which shows physical behaviour. The model definition may be derived either from analytic solutions to the underlying physics or from phenomenological considerations. Applicability is limited to the validity of the simplification process. Non-physical behaviour may occur otherwise.

Example: transmission line model for fluid flow in a pipe.

L2. Models are based on balance equations. These equations are spatially averaged over the component. The models show a correct physical behaviour unless the assumptions for the averaging process are violated.

¹This results in a flat model hierarchy and restricts the use of inheritance.

Example: single control volume for fluid flow in a pipe.

L3. Models are by construction subdivided into a fixed number of spatial zones. The spatial localisation of these zones is not necessarily fixed and can vary dynamically. For each zone a set of balance equations is used and the model properties (e.g. media data) are averaged zone-wise. The models show a correct physical behaviour unless the assumptions for the zonal subdivision and the averaging process over zones are violated.

Example: moving boundary approach for fluid flow in a pipe.

L4. Models can be subdivided into an arbitrary number of spatial zones (control volumes) by the user. They thus provide a true spatial resolution. For each zone a set of balance equations is used which is averaged over that zone. The model shows a correct physical behaviour unless the assumptions for the choice of grid and the averaging process over the control volumes are violated.

Example: finite volume approach with spatial discretisation in flow direction for fluid flow in a pipe.

3.2.2 Physical Effects to be Considered

Once the decision for a specific detail group of models is made, the set of required physical effects to be covered by a model may still differ according to the simulation goal. For instance, in a pipe model it might be necessary to resolve the spatial flow properties but unnecessary to analyse sound waves in detail. This is reflected in the complexity of the basic physical equations underlying the model.

Notice that, although the ClaRaCCS library is designed for dynamic simulations, it is still possible to include models, where parts of the basic physical equations correspond to the stationary behaviour of a component. Such models are often favourable with respect to computation time and stability. Their use is appropriate whenever certain aspects of the component dynamics can be neglected compared to the system dynamics under consideration. In the pipe example above this would be manifested by the fact that if only fluid flow properties (temperature profile, flow velocities, etc.) are of interest, sound wave propagation can be neglected, as long as the flow velocity is much less than the speed of sound. Consequently a stationary momentum balance for the fluid would be sufficient in this case.

In order to cope with these different needs, the ClaRaCCS library provides *component models at the same level of detail but covering different physical effects*. They are distinguished by different self-explaining names.

3.2.3 Level of Insight

By now, the fundamental equations of a model are defined by setting its level of detail and the physical effects of consideration. However, these equations declare *which* physical effects are considered, but not *how* they are considered. For instance, the pressure loss in a pipe may be modelled using constant nominal values or via correlations taking the flow regime and the fluid states into account. These physical effects are therefore modelled in *replaceable* models that complete the fundamental equations using predefined interfaces, e.g. the friction term in the momentum balance. By separating the governing model definition from the underlying sub-models, the flexibility of the model is enhanced without losing readability.

3.3 Media Data

The property data for all models will be provided by medium classes which then in turn call external C-functions. However, up to now only the models of the conventional part of the plant obtain their property data from external functions. The observed advantage of this procedure is the possible access to other commercial external fluid property libraries and a large increase in simulation speed of the models. The implementation of external property data for the multi-component media used in the PCC is still work in progress. The current state of this issue and the experiences with external, table-based media data for single-component media are very encouraging concerning simulation speed and simulation stability.

For the sake of initialisation and numerical stability the choice of different state variables may be of high importance. Depending on the selection of the respective state variables an index reduction can be necessary. Also phase and reaction equilibria can lead to high index systems which have to be reduced symbolically. In both cases it is likely that derivatives of property data are required to perform index reduction. Providing these derivatives still is a challenge to be overcome during this project.

3.4 Model Validation

Models in the ClaRaCCS library will be validated against established process modelling and power plant software (Aspen Plus and Epsilon [11, 12]) as well as dynamic measurement data.

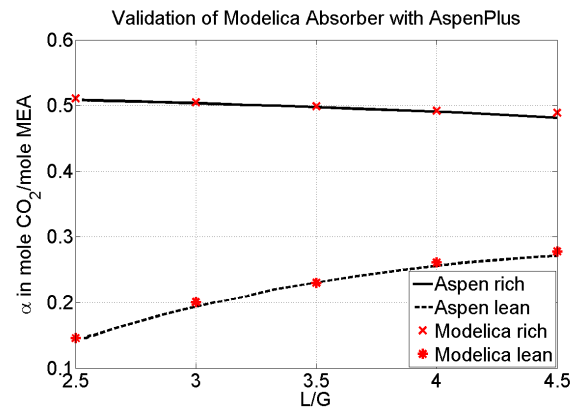


Figure 4: Validation example. Modelica absorber vs. [11]. Lean and rich solution. Here α denotes the amount of substance CO_2 per amount of substance solvent (MEA). L denotes the mass flow rate of the solvent and G is the flue gas mass flow rate.

For the water steam cycle this measurement data are provided by the coal fired power plant [13] at Rostock, Germany, with a net power output of 500 MW. For the PCC process the data are provided by a pilot plant [14] at Heilbronn, Germany, which has the capacity to clean approximately $1150 \text{ Nm}^3/\text{h}$ of flue gas.

4 Modelling of Furnace

The purpose of this section is the illustration of the general modelling strategy as introduced in section 3. As an example the furnace model package that provides models for burner, flame rooms and hoppers with different levels of detail is considered. Here, predominantly the structure of the package is described without covering the physics inside in more detail.

4.1 Connectors

Although parts of ClaRaCCS and Modelica.Fluid cover similar fields of application own connectors for liquids, fluids and gas mixtures are necessary due to the usage of external media as motivated in 3.3. However, models of the Modelica standard library and ClaRaCCS may be connected using simple adapters included in the library. In addition, connectors for

aerosols (unburned coal dust, fly ash) are defined similar to the approach of Gall *et al.* [15]. These connectors instantiate the connectors for the flue gas, the coal and the slag. In contrast to Gall's approach, the fly ash is treated as substance of the flue gas so that there is no need for a fourth connector. The Modelica code of e.g. the CoalSlagFlueGas_inlet connector reads

```
connector CoalSlagFlueGas_inlet
"Port describing Coal,Slag and FlueGas flow"
import ClaRaCCS;
// Media properties of coal and slag
parameter Media.Coal.PartialCoal coalType;
parameter Media.Coal.PartialSlag slagType;

BaseClasses.Interfaces.FlueGas_a flueGas;
BaseClasses.Interfaces.Coal_inlet
    coal(coalType = coalType);
BaseClasses.Interfaces.Slag_outlet
    slag(slagType = slagType);
end CoalSlagFlueGas_inlet;
```

Likewise, a connector for the coal dust and the primary air is available. In addition, components for splitting and joining are provided so that other components in the flue gas path having solely flue gas connectors (as e.g. a deNO_x plant) can be connected to a combustion chamber model.

4.2 Components in the Furnace Package

4.2.1 A Simplified Combustion Chamber Model

Figure 5 shows the tree of the furnace model package. At the top level it provides the model SimpleCombustionChamber which represents a simplified model of detail level 2 (refer to figure 6). It provides physical connectors for the coal dust and primary air, for the slag and flue gas outlet. Based on a stationary stoichiometric combustion calculation, the flue gas composition, the heat $Q_{\text{combustion}}$ obtained from combustion and the stoichiometric air ratio $\lambda = \dot{m}_{\text{air}}/\dot{m}_{\text{air,st}}$ are calculated. The model consists of stationary balance equations for the energy, the mass flow, the flue gas components and its composition (i.e. mass balance equations for each single substance considered in the used flue gas mixture). The user can

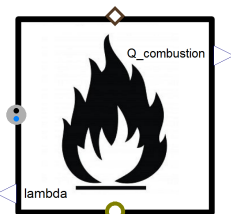


Figure 6: Diagram view of the simplified combustion chamber model.

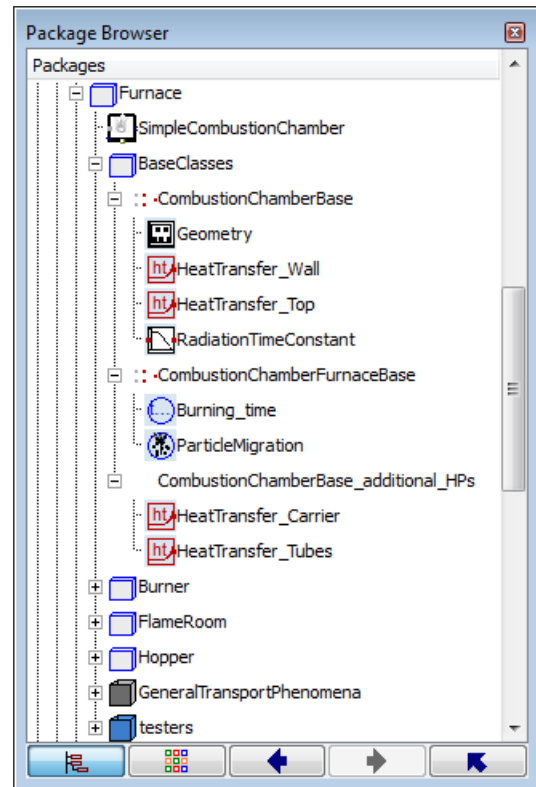


Figure 5: The tree of the furnace model package.

set values for the flue gas outlet temperature, the slag fraction, slag temperature, and the concentration of toxic substances (CO, NO_x and SO_x) in the flue gas. Whereas the flue gas and the slag are accessible via physical connectors, $Q_{\text{combustion}}$ and λ are provided by *real outputs*. These important process variables can then be used as inputs to other models, such as a controller for the air ratio. The model has been compared to Epsilon [12] and has shown good consistency of the results.

4.2.2 Components for a Detailed Combustion Chamber Model

Besides the model SimpleCombustionChamber, the furnace package is intended to provide all required components from that a complete –more detailed– combustion chamber model can be built. These components are currently models of detail level 2, i.e. they represent single control volumes for the considered combustion process. Since a complete combustion chamber model will be built from several level of detail 2 models, it yields a spatial discretisation and will thus be a model of detail 3 or 4.

Figure 7 shows the diagram of a burner model which extends the three base models– namely CombustionChamberBase,

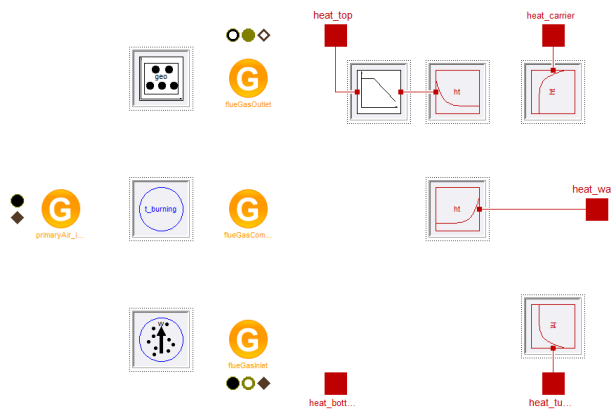


Figure 7: Diagram view of a burner model.

CombustionChamberFurnaceBase and CombustionChamberBase_additional_HPs that are provided by the BaseClasses package (see figure 5).

The partial model CombustionChamberBase provides the definition of the used Media, the instance of the corresponding medium objects and physical connectors. A *replaceable model* Geometry allows for an adaptation of the combustion chamber's dimensions to the user's needs. Also basic parameters that are common in all furnace components are defined in this base class whereby the duplication of code is avoided ensuring low maintenance effort.

Regarding the physical effects to be considered, besides the connectors for the gas and solid flow, this base model has three connectors for heat flows. They are required to model the heat transfer from the hot flue gas to the combustion chamber wall and the heat transfer between neighboured flame rooms/burners. The heat flows to the top and to the wall are calculated based on the *replaceable models* HeatTransfer_Top and HeatTransfer_Wall (refer to figure 5). Please note that the heat flow at the bottom connector is calculated from the heat transfer model HeatTransfer_Top in the respective adjacent burner/flame room. In view of a numerical optimisation, the control volume temperature can be decoupled from that of neighboured ones by using differential states for the temperature at a heat port. Such a state with a certain time constant is provided with the *replaceable model* RadiationTimeConstant (see figure 5) and is placed in the burner model shown in figure 7 at the top heat connector. Whereas the CombustionChamberBase represents a basic control volume just describing the flow of the gas and solid phase and the heat transfer, the sec-

ond partial model CombustionChamberFurnaceBase accounts for the furnace process. It is extended by *replaceable models* for the burning time and the particle migration time. The third base model CombustionChamberBase_additional_HPs provides two additional heat ports. In this way also the heat flow from the flue gas to e.g. the carrier tubes and the tube bundles of the convective heat exchangers in a boiler model can be modelled. Again, for each heat port replaceable models for the heat transfer correlation are provided.

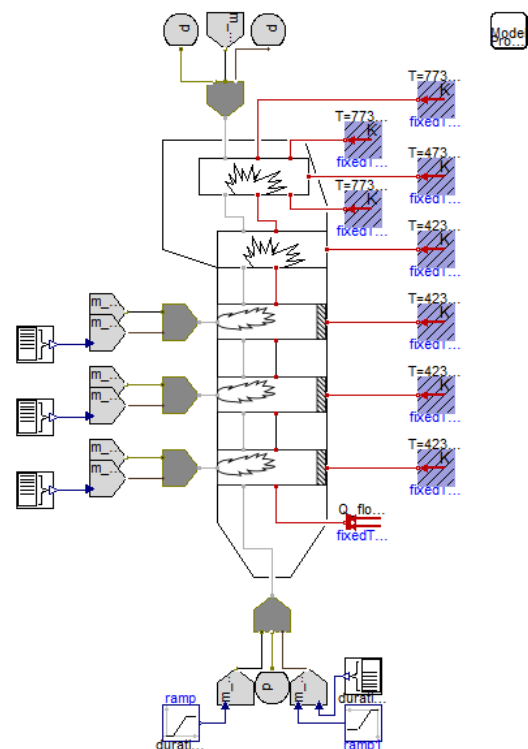


Figure 8: Diagram view of an exemplary combustion chamber model built by the furnace package components described above.

Figure 8 shows the diagram of an exemplary combustion chamber model. In a boiler model the fixed temperature boundaries on the right would be replaced by according water steam tube models.

5 Example of Use

The current capabilities of the ClRaCCS library can be illustrated by a model of an anthracite-fired steam power plant with a coupled post combustion capture unit. For the sake of simplicity and due to current library limitations the complex topology of current Rankine cycles is reduced to the main features.

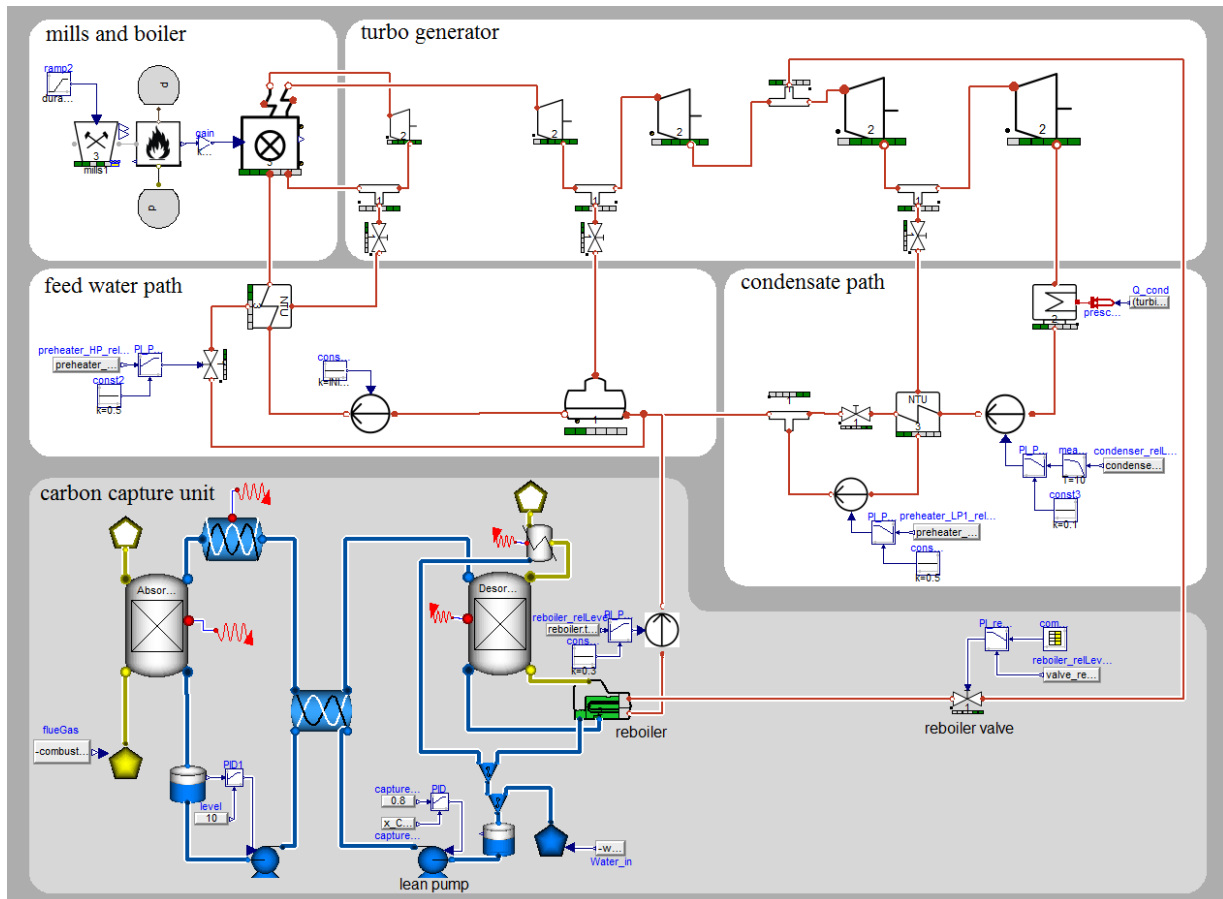


Figure 9: Diagram view of coupled steam plant with carbon capture unit.

5.1 Example Description

The model's definition is based on a PCC-retrofit of the existing power plant [13] of Rostock, Germany, see table 1 for its general operation parameters.

Table 1: General overview of power plant of Rostock, Germany

Net output	509 MW
Net efficiency	43.2 %
Live steam pressure	262 bar
Live steam temperature	545 °C
Live steam mass flow	417 kg/s
Re-heat temperature	562 °C
Re-heat pressure	54 bar

In particular, the model features a set of roller bowl mills, as reported in [16], a reduced combustion chamber and boiler, a turbo-generator with tapplings for one high pressure pre-heater, one low pressure pre-heater, the feedwater tank and the reboiler of the PCC. At the low pressure side a condenser, a condensate pump, the pre-heaters, the feedwater tank and the feedwater pump complete the cycle, see figure 9.

The coupling of furnace outlet and PCC inlet at the flue gas path is currently cut because of missing components for the flue gas cleaning.

The PCC features first-principle models for the absorber and desorber columns and simplified models for pumps and the inner heat exchanger, see [17] for a more detailed description on column modelling.

In order to get a pure feed-forward response of the model, only subordinate controllers are implemented in a simple way. Pumps are used to keep the filling levels of the storage devices within reasonable bounds. The generator power output is controlled by the reboiler valve, which sets the amount of steam that is used to supply heat for the reboiler. Additionally the carbon capture rate is controlled by the lean solvent pump downstream the desorber. Future investigations will have to consider an integrated unit control concept for both the steam cycle and the PCC unit, see [18] for a first approach.

Although the degree of simplification is too high to allow quantitative statements on the transient behaviour, the model is capable to capture the main dynamics in a qualitative manner and shows that the dif-

ferent aspects of the library work together as desired.

5.2 Simulation Results

The extensive steam tapping for the heating of the reboiler introduces the option to provide primary control power by throttling the reboiler valve. Doing so, the low pressure turbine mass flow rate is increased in short term resulting in a significant power step-up. However, a temporary drop of the carbon capture rate has to be accepted. In figure 10 the power output and the reboiler steam mass flow are displayed indicating that almost full throttling of the valve can lead to a power step of 5 %-pts within 30 s. After holding the primary control power for 5 min, the control band is set free within 10 min.

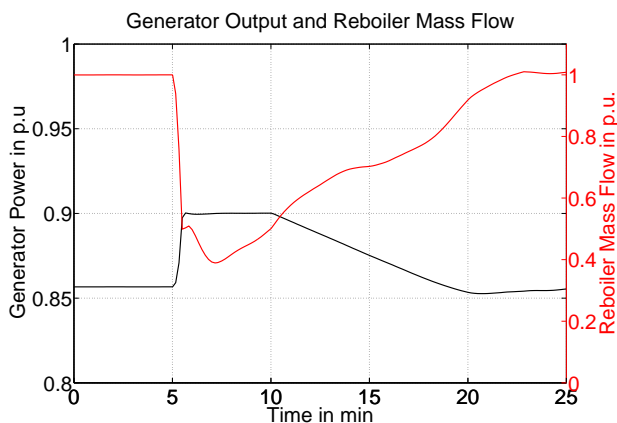


Figure 10: Power output applying reboiler feed reduction.

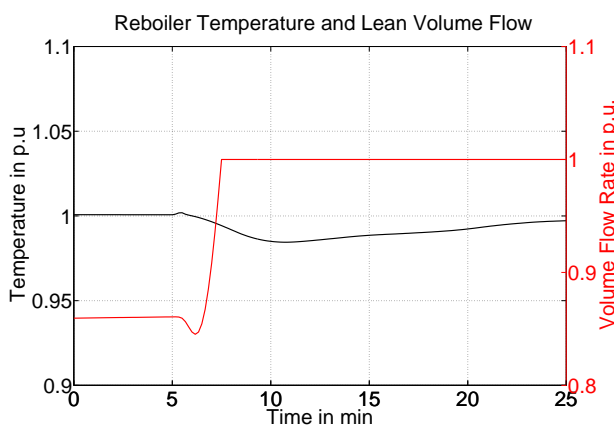


Figure 11: Amine gas treatment characteristic values.

The reboiler temperature in figure 11 shows a moderate drop with a minimum at 10 min of simulation time. Due to the throttling of the valve and the subsequent temperature drop in the reboiler, the lean loading of the solution in the reboiler increases (because less

CO₂ is stripped from the solution). This means that a higher flow rate of solution is needed to maintain the targeted capture rate. Hence, the lean pump volume flow increases in order to compensate the higher loading, until the pump reaches its maximum capacity.

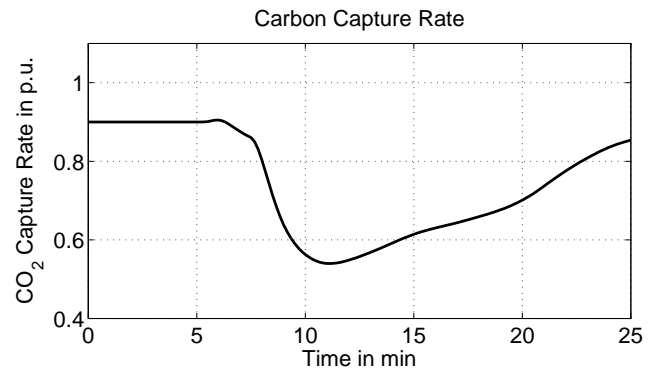


Figure 12: Carbon capture rate during reboiler hold-up.

As expected, the additional power output comes at the cost of a strongly reduced carbon capture rate showing its minimum of 55 % at approx. 11 min simulation time, see figure 12.

Although the simulation scenario might be strongly simplified and the applied control strategy technically not yet mature, it becomes obvious that the application of highly integrated sub-processes like the amine gas-treatment brings up new options for the plant's transient operation mode and economical shifting (trade-off between revenues from primary control power supply and costs due to CO₂ certificates) on the one hand. On the other hand new challenges for the power plant's control system must be tackled.

6 Summary and Outlook

In this paper the status of development of the C1aRaCCS library is presented, which is a central part of the DYNCAP project. In its final stage the library will allow detailed dynamic simulations of power plants coupled to a CO₂ capturing process.

It was demonstrated how the flexible library structure supports the user in order to build up complex power plant models individually tailored to specific simulation goals.

Although in an early state of development, the given simulation example proves that the library is already capable of simulating simplified dynamic operation scenarios for coal fired power plants coupled to a post-combustion CO₂ capture process.

Having almost completed the development of fundamental components for the water steam cycle and the post combustion process, the development will now proceed to the design of subsystems and complete power plant models including a CO₂ capturing unit. These models will be validated against measurement data from an existing hard coal power plant and a demonstration post combustion unit.

However, it should be noted that the design of ClaRaCCS allows the easy adaptation of component models in order to feature CCS-retrofits to existing power plants as well as to perform concept studies for planned ones. In this context the development of an integrated control concept is a major challenge. First steps into this direction have already been published in [18]. Moreover the automation of the initialisation process for complex simulations will be a major future direction of work.

Concerning the models for the CO₂ capture, the use of external media data shall be supported in the future. In this context it may be necessary to adapt the models in order to maintain performance.

7 Acknowledgements

On behalf of the authors we would like to thank all members of the ClaRaCCS team. This research project is supported by the Federal Ministry of Economics and Technology (project number 03ET2009). For valuable data input and discussions the staff of KNG, EnBW, Vattenfall and E.ON are gratefully acknowledged. We thank the anonymous reviewers for their valuable comments.



References

- [1] B. Metz, O.R. Davidson, P.R. Bosch, and R. Dave. *Climate Change 2007: Mitigation. Contribution of Working Group III to the Fourth Assessment Report of the Intergovernmental Panel on Climate Change*. IPCC, Cambridge, United Kingdom and New York, NY, USA., 2007.
- [2] COORETEC. <http://www.cooretec.de/index.php?index=21> (retrieved 09th May, 2012).
- [3] Dyncap Project, 2011-2014. <http://www.kraftwerkforschung.info/en/mehr-flexibilitaet-fuer-emissionsarmekohlekraftwerke> (retrieved 10th May, 2012).
- [4] Dymola. Dassault Systèmes, 2012.
- [5] SimulationX®. ITI Gesellschaft für Ingenieurtechnische Informationsverarbeitung mbH, 2012.
- [6] A. Kather, S. Rafailidis, C. Hermsdorf, M. Klostermann, A. Maschmann, K. Mieske, J. Oexmann, I. Pfaff, K. Rohloff, and J. Wilken. *Research & development needs for clean coal deployment*. Number CCC/130 in ISBN 978-92-9029-449-3. IEA Clean Coal Centre, January 2008.
- [7] J. Oexmann. *Post-Combustion CO₂ Capture: Energetic Evaluation of Chemical Absorption Processes in Coal-Fired Steam Power Plants*. PhD thesis, University Hamburg-Harburg, Institute of Energy Systems, Hamburg, January 2011. ISBN 978-3-86955-633-8.
- [8] G. T. Rochelle. Amine Scrubbing for CO₂ Capture. *Science*, 325:1652–1654, 2009.
- [9] A. Kather and G. Scheffknecht. The oxycoal process with cryogenic oxygen supply. *Naturwissenschaften*, 96(9):993 – 1010, 2009.
- [10] M. Bonvini and A. Leva. Scalable-detail modular models for simulation studies on energy efficiency. In *Proceedings 8th Modelica Conference, Dresden, Germany, March 20-22, 2011*, 2011.
- [11] AspenPlus®. Aspen Technology, Inc., 2011.
- [12] EBSILON® Professional. Evonik Energy Services GmbH, 2011.
- [13] KNG Power Plant Rostock. <http://www.kraftwerk-rostock.de> (retrieved 09th May, 2012).
- [14] EnBW Energy. 2010 Innovation Report, 2010.
- [15] L. Gall, K. Link, and H. Steuer. Modeling of gas-particle-flow and heat radiation in steam power plants. *Modelica Conference, Dresden, Germany, March 20-22, 2011*.
- [16] P. Niemczyk, P. Andersen, J.D. Bendtsen, T.S. Pedersen, and A.P. Ravn. Derivation and validation of a coal mill model for control. In *IFAC Symposium on Power Plants and Power Systems Control 2009, Tampere*, July 2009.
- [17] K. Dietl, A. Joos, and G. Schmitz. Dynamic analysis of the absorption/desorption loop of a carbon capture plant using an object-oriented approach. *Chemical Engineering and Processing: Process Intensification*, 52:132 – 139, 2011.
- [18] F. Gottelt, K. Wellner, V. Roeder, J. Brunnemann, G. Schmitz, and A. Kather. A Unified Control Scheme for Coal-Fired Power Plants with Integrated Post Combustion CO₂ Capture. In *8th IFAC Conference on Power Plant & Power System Control, Toulouse*, 2012. accepted for publication.

Start-up Optimization of a Combined Cycle Power Plant

A. Lind^a, E. Sällberg^a,
S. Velut^b, S. Gallardo Yances^c, J. Åkesson^{a b}, K. Link^c
^aLund University, Department of Automatic Control, Lund, Sweden
^bModelon AB, Lund, Sweden
^cSiemens AG, Energy Sector, Erlangen, Germany

Abstract

In the electricity market of today, with increasing demand for electricity production on short notice, the combined cycle power plant stands high regarding fast start-ups and efficiency. In this paper, it has been shown how the dynamic start-up procedure of a combined cycle power plant can be optimized using direct collocation methods, proposing a way to minimize the start-up time while maximizing the power production during start-up. Physical models derived from first principles have been developed in Modelica specifically for optimization purposes, in that the models contain no discontinuities. Also, the models used for optimization are simpler than typical high-fidelity simulation models. Two different models used for optimization in four different start-up scenarios are presented in the paper. A critically limiting factor during start-up is the stress of important components, e.g., the evaporator. In order to take this aspect into account, constraints on the stress levels of such components have been introduced in the optimization formulation. In particular, it is shown how a pressure dependent stress constraint, similar to what is used in actual operation, can be applied in optimization. Also, different assumptions about which control variables to optimize are explored. Results are encouraging and show that energy production during start-up can be significantly increased by increasing the number of control inputs available to the optimizer, while maintaining desirable lifetime of critical components by introducing constraints on acceptable stress levels.

Keywords: Combined Cycle Power Plants, Start-up, Dynamic optimization, Optimica, Control, Modelica, Modeling

1 Introduction

In a time when the production from renewable energy sources is steadily growing the demand for complementary electricity production on short notice is high. Large fluctuations during the day require power generators to react quickly to maintain the balance between demand and production. Deregulation of the electric power market also allows private investors to install power plants and supply power to the grid, which has increased the competition on the electricity market. The requirements between demand and supply have to be maintained while offering electricity at the lowest cost.

When considering fast start-ups and efficiency, the combined cycle power plant stands high in comparison with other electricity production methods. In this paper, the start-up procedure of a combined cycle power plant is studied. The aim is to minimize the start-up time while keeping the lifetime consumption of crucial power plant components under control and maximizing the amount of power output produced.

Several previous studies that deal with optimization of the start-up of combined cycle power plants have been made. In Casella and Pretolani, [1], optimization with a trial-and-error method is presented where the results are obtained by simulating Modelica power plant models. The study has been carried out to develop simplified models that can be used to automatically compute the optimal transients with an optimization software and the models were based on the Modelica ThermoPower library, see Casella and Leva, [2]. A model-based approach for optimizing the gas turbine load trajectory has been studied in Casella et al., [3]. A simplified model is developed based on interpolated locally identified linear models and the procedure aims at deriving the gas turbine load profile described by a parameterized function. A minimum-time problem is solved to determine the parameters

of the parameterized function. In [4] a combined cycle power plant is modeled and optimized, where the thermo-mechanical stress in the steam turbine rotor is considered as the most limiting factor during the start-up. Shirakawa et al. proposed an optimal design method combining dynamic simulation and nonlinear programming in [5].

The aim of the current paper is to make the start-up procedure of a combined cycle power plant more efficient, with respect to the start-up time and power production, while limiting the thermal stress in the heat recovery steam generator. The plant models are described in the object-oriented modeling language Modelica. All models are developed by Siemens AG, Energy Sector, in cooperation with Modelon AB, and are based on elementary models from first principle equations of mass and energy. The physical models have been developed using the commercial Modelica simulation environment, Dymola [6] and they have been adapted to suit optimization purposes. The tool used for optimization is the Modelica based open source platform *JModelica.org*.

The paper is structured as follows: Section 2 gives some background information about combined cycle power plants, dynamic optimization, JModelica.org and Optimica, while Section 3 describes the power plant model. Section 4 presents the optimal start-up problem formulation and the numerical results are discussed. Section 5 summarizes the results of this paper and gives proposals for future work.

2 Background

2.1 Combined Cycle Power Plants

The basic principle of a combined cycle power plant (CCPP) is to combine two thermal cycles in one power plant, where the topping cycle is a cycle operating at a higher temperature and the bottoming cycle is a cycle operating at a lower temperature level. The waste heat that the topping cycle produces is used in the process of the bottoming cycle and the efficiency is higher for the combined cycle than that of one cycle alone. In the commercial power generation of today the combined cycle power plants consist of a gas topping cycle and a steam/water bottoming cycle [7].

The plant is constructed mainly with three parts, the gas turbine (GT), the heat recovery steam generator (HRSG) and the steam turbine (ST).

In the GT, ambient air is drawn into the turbine, compressed and used to burn some combustion

medium. Hot gas is produced and expands in the turbine where it is used to drive both the compressor and the generator.

The key component of a CCPP is the HRSG which couples the two cycles so that the heat from the GT exhaust gas is used to produce hot steam which drives the ST. The HRSG consists mainly of three components; the economizer, the evaporator and the superheater. The water is preheated in the economizer, evaporated to wet steam in the evaporator and the steam is dried in the superheater. When the steam is of high enough quality it is expanded in the ST where it generates power.

The net efficiency can reach more than 60% in today's CCPPs. About 60-70 % of the total power output is produced in the GT [7].

The start-up of a CCPP is normally scheduled as follows:

- 1 The GT is first accelerated to full speed no load and it is synchronized to the grid.
- 2 The load of the GT is increased and the boiler starts producing steam. The generated steam is not led to the ST but bypassed to a condenser.
- 3 When the steam quality is high enough, the bypass valve is slowly closed and the steam can drive the ST.

Reducing the start-up time of the CCPP is typically achieved by maximizing the loading rates of both turbines while maintaining the lifetime consumption of critically stressed components under control. One of the critical components is the drum in the evaporator. During the second phase of the start-up, the walls of this component are subject to high thermal stress due to temperature gradient transients. The ST is also subject to large stress constraints, but this occurs in the last phase of the start-up. The paper focuses on the optimization of the second phase, that is the loading of the GT.

2.2 The Dynamic Optimization Problem

The start-up optimization of the CCPP has been formulated as a dynamic optimization problem. The optimization consists typically in finding time trajectories of the control variables, $u(t)$, that minimize an objective function φ expressed in terms of process variables y . The optimization problem can generally be stated as:

$$\min_{u(t)} \varphi(z(t), y(t), u(t), t_f) \quad (1)$$

subject to

$$\frac{dz(t)}{dt} = F(z(t), y(t), u(t), t) \quad (2)$$

$$0 = G(z(t), y(t), u(t), t) \quad (3)$$

$$z(0) = z^0 \quad (4)$$

with the bounds

$$z^L \leq z(t) \leq z^U \quad (5)$$

$$y^L \leq y(t) \leq y^U \quad (6)$$

$$u^L \leq u(t) \leq u^U \quad (7)$$

$$t_f^L \leq t_f \leq t_f^U \quad (8)$$

where

φ is a scalar objective function,

F are the right hand sides of differential equation constraints,

G are algebraic equation constraints, assumed to be index one,

z are differential state profile vectors,

z_0 are the initial values of z ,

y are algebraic state profile vectors,

u are control profile vectors,

t_f is the final time. [8]

The objective function φ , that is to be minimized, can have multiple forms; one is given by the *Lagrange* form:

$$\varphi = \int_{t_0}^{t_f} L(z(t), y(t), u(t), t) dt. \quad (9)$$

2.3 JModelica.org

In this project, the tool used for optimization is the open source platform JModelica.org [9].

JModelica.org is an extensible Modelica-based open source platform for optimization, simulation and analysis of complex dynamic systems. The main objective of the project is to create an industrially viable open source platform for optimization of Modelica models, while offering a flexible platform serving as a virtual lab for algorithm development and research. [9]

JModelica.org offers different types of model objects that can be used for simulation and optimization. For simulation purposes, a Functional Mock-up Unit (FMU) that follows the FMI (Functional Mock-up Interface) standard, is used. It is created by compiling a Modelica model in JModelica.org or in any other tool which supports the FMU export. The FMU file

is thereafter loaded as an FMUModel Python object in JModelica.org and can be simulated using the *Assimulo* package. For a more detailed description of import and export of FMUs in python, see [10]. For optimization purposes a JMUModel object is instead created. A JMU is a compressed file following a JModelica.org specific standard that is close to the FMI standard. After compilation, the JMU file is loaded into JModelica.org and the JMUModel is created and can be optimized using state of the art numerical methods. See Åkesson et al. [11] for a thorough description of the JModelica.org platform.

2.3.1 Collocation Method

The JModelica.org platform uses a direct collocation method based on Lagrange polynomials on finite elements with Radau points [12]. The Differential Algebraic Equations (DAE) are transformed to a nonlinear program (NLP) by approximating control and state profiles by piecewise polynomial. The NLP problem is solved by the solver IPOPT [8].

2.3.2 IPOPT

The open-source software IPOPT (Interior Point Optimizer) is a package for large-scale nonlinear optimization. The optimization problem is transferred to an interior point problem formulation where a logarithmic barrier term replaces the inequality constraints [13].

2.4 Optimica

Optimica is an extension of the Modelica language that enables high-level formulation of optimization problems based on Modelica models. The extension mainly consists of an additional class, *optimization*, which includes the attribute *objective* that specifies the objective function of the optimization problem. Another supplement is the *constraint* section, which can handle different kinds of linear and non-linear equality- and inequality constraints. [14]

3 Models

3.1 Plant Model

In this paper, three models of a CCPP with different complexities have been considered referred to as CCPP1, CCPP2 and CCPP3, see Figures 1, 2 and 3, respectively.

All models are developed in Modelica, [15], using the commercial modeling and simulation environment Dymola [6] and are based on elementary models from first principle equations of mass and energy. Discontinuities have been smoothed and all equations are twice continuously differentiable. Components are modeled separately according to the object-oriented principle and joined by additional connection equations to form the complete system model. Some of the components in the Dymola models are not connected by visible connector lines but only by Modelica equations. This is the case for the output of the integrator at the valve opening, which is connected to the real expression at the valve just above it and also the two outputs of the GT which are connected to the two real expressions to the right of the GT.

The water side is modeled by dynamic balance equations whereas the gas side is static. The simplified HRSG model, see Figures 1, 2 and 3, consist of an HP pressure stage boiler and is represented by lumped volume models of a superheater and an evaporator. To attain better accuracy with respect to thermal discretization, the superheater is described by five partial components with different tube geometries. An ideal level control is assumed in the evaporator model and it computes the water/steam flow through the HRSG. The evaporator drum is modeled as a volume, where the wall, which is subject to high stress during transients, is spatially discretized. The GT model computes temperature and mass flow of the gas entering the HRSG at every load. The bypass valve controls the pressure in the water circuit and can be actuated by a pressure controller to limit large pressure transients. A constant pressure has been chosen as boundary condition for the bypass valve, corresponding to the pressure in the condenser. The models CCPP1 and CCPP2 differ in that a pressure controller acting on the bypass valve is introduced in CCPP1, whereas in CCPP2, the bypass valve is used as a manipulated control variables available for optimization.

The model CCPP3, see Figure 3 is more detailed than models CCPP1 and CCPP2 in that it is modeled with an additional IP reheater apart from an HP superheater and an HP evaporator. The reheater is described by three partial components and the superheater has four partial components with different tube geometries which are operating at different pressures like in CCPP1 and CCPP2 as in the simplified model. An additional component that has been added to CCPP3 is the header of the part of the superheater operating at highest temperature, see component *Header* in Figure

3. The header is in this model considered as a component subject to high stress during start-up transients together with the evaporator drum.

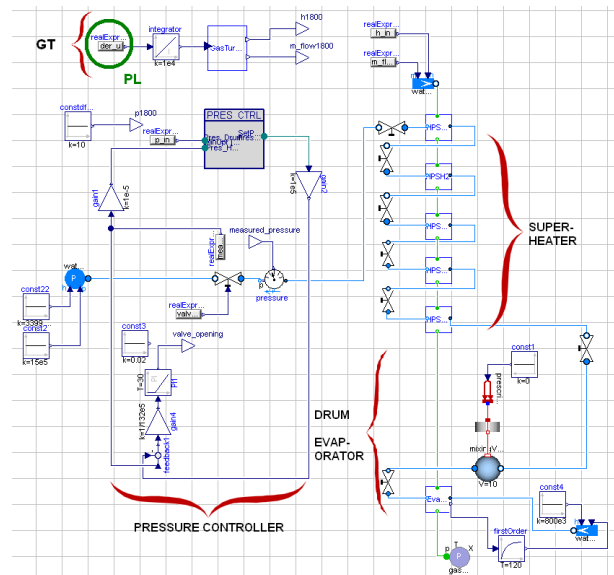


Figure 1: Modelica object diagram of model CCPP1, including a pressure controller. The main components are marked and the degree of freedom PL (Power Load) is circled.

3.2 Water and Steam Properties

Pressure and specific enthalpy have been chosen as states in the balance equations on the water side. Correlations to compute temperature as well as density and its derivatives with respect to pressure and enthalpy need therefore to be derived. Polynomial approximations expressed as Taylor expansions from the phase boundaries have been chosen, see [16] for a similar method. This leads to optimization friendly and accurate medium properties and also a continuous transition of temperature and density across the phase boundaries.

4 GT Load Profile Optimization

4.1 Problem Formulation

The aim of the optimization is to minimize the start-up time of the CCPP while keeping the lifetime consumption of critically stressed components under control and maximizing the amount of power output produced. Four different optimization problems are considered, namely, *i)* a 1DOF problem based on the model CCPP1 is considered, *ii)* a 2DOF problem

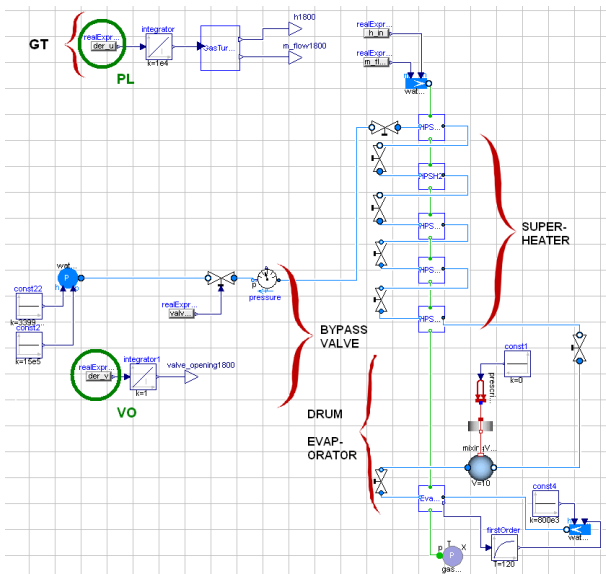


Figure 2: Modelica object diagram of model CCPP2. The main components are marked and the degrees of freedom PL (Power Load) and VO (Valve Opening) are circled.

based on CCPP2, *iii*) a 2DOF problem with constant thermal stress bounds based on CCPP3 and finally *iv*) a 2DOF problem with pressure dependent thermal stress bounds based on CCPP3.

4.1.1 Optimization Phase

The optimization starts after the synchronization of the GT to the grid. The time between stand-still and full-speed-no-load is not subject to optimization but is simulated to compute the initial point of the optimization. In the present study, a hot start is assumed, which means that the start-up is initiated after a stand-still time of at most 7 hours. The start-up is considered to be complete when the GT has reached its full load, i.e. its maximum power output.

4.1.2 Degrees of Freedom

Two control variables have been considered in the paper: the load u of the GT and the opening v of the bypass valve. The degrees of freedom in the optimization are defined as the time-derivative of the control variables, i.e. du/dt (marked as PL for Power Load in Figures 1, 2 and 3) and dv/dt (marked as VO for Valve Opening in Figures 2 and 3), and are parameterized by piecewise constant signals.

In a first optimization problem, the GT load u is chosen to be the only control variable. The bypass valve is in that case manipulated by a PI controller to con-

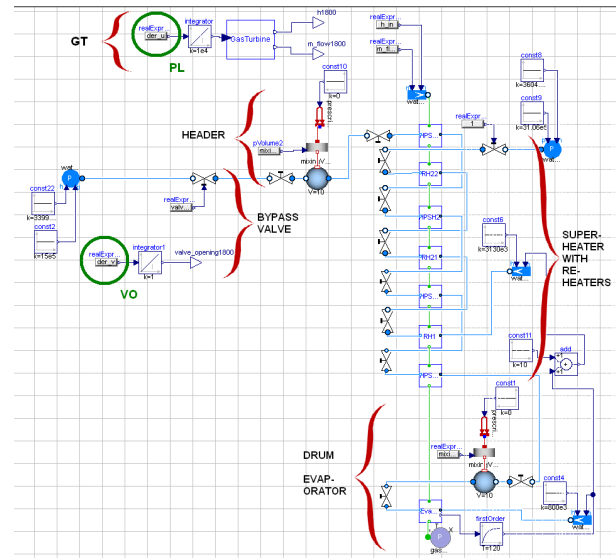


Figure 3: Modelica object diagram of model CCPP3, including a header and an IP reheater. The main components are marked and the degrees of freedom PL (Power Load) and VO (Valve Opening) are circled.

trol the pressure at the superheater outlet, Figures 1. In a second optimization problem, the pressure controller, seen in Figure 1, is removed and both degrees of freedom are used for optimization. In these cases, the physical models of the power plant are identical, apart from the pressure controller. For the two optimization problems based on CCPP3, both degrees of freedom are used.

4.1.3 Cost Function

The objective function is written in the *Lagrange* form as in Equation (9). The optimization problem has been formulated using a quadratic cost function where the integrand L penalizes the deviation of the load u from its reference value u_{ref} as well as the derivatives of the inputs:

$$L = \alpha(u - u_{ref})^2 + \beta \frac{du^2}{dt} + \gamma \frac{dv^2}{dt}. \quad (10)$$

The reference value for $u(t)$ is normalized to 1, which corresponds to 100% of its full load. This formulation maximizes the produced power output during start-up and should also result in a short start-up time.

4.1.4 Constraints

The limiting factor during the start-up procedure is the thermal stress due to temperature gradient transients in the wall of the drum of the evaporator and superheater header. The simplified optimization constraint

considered for CCPP1 and CCPP2 is the temperature gradient in the wall of the boiler:

$$|T_{middle\ layer\ wall\ Drum} - T_{evap}| \leq |dT_{max\ Drum}| = 0.5. \quad (11)$$

In the first optimization problem based on CCPP3, an additional constraint for the temperature gradient in the header is added:

$$|T_{middle\ layer\ wall\ Header} - T_{SH}| \leq |dT_{max\ Header}| = 0.5. \quad (12)$$

In the second optimization problem based on CCPP3, the constant bounds on the drum and header temperature gradients are replaced by pressure dependent constraints

$$|T_{middle\ layer\ wall\ Drum} - T_{evap}| \leq f_d(p) \quad (13)$$

$$|T_{middle\ layer\ wall\ Header} - T_{SH}| \leq f_h(p). \quad (14)$$

When the bypass valve opening is used for optimization, an additional constraint on the opening derivative is introduced:

$$\left| \frac{dv}{dt} \right| < \left| \frac{dv}{dt} \right|_{max}. \quad (15)$$

4.1.5 Initialization

To initialize the first optimization problem, a simulation of the model is first realized in JModelica.org, using a simple (zero-load) input trajectory. This results in feasible trajectories for the optimization that do not violate the defined constraints. The simulation result is then used as an initial guess trajectory for the first optimization. To improve the result accuracy the optimization is done iteratively, starting with a simple discretization with few elements. The result of the previous optimization is then used as a new initial guess trajectory and the discretization is refined by increasing the number of elements and/or by changing the end time of the optimization.

4.1.6 Optimization Settings

The number of elements, n_e , in the optimization interval has been varied between 10 and 45 and the number of collocation points in every element was fixed to $n_{cp} = 3$. The overall relative tolerance for the interior point solver was chosen to be 10^{-4} .

4.2 1 DOF Optimization of CCPP1

The pressure in the HRSG is controlled using the opening v of the bypass valve in a built in control loop, leaving u as the sole degree of freedom (1

DOF) for optimization. The continuous-time optimization model contains 28 continuous time states and 456 scalar equations. The power output has been allowed to either both increase and decrease during start-up (non-monotonic power output) or to only increase (monotonically increasing power output). Both cases have been optimally controlled to full load and the optimization results are shown in Figures 4 and 5. The solid line trajectory represents the solution for the non-monotonic power output and the dashed trajectory represents the monotonically increasing power output.

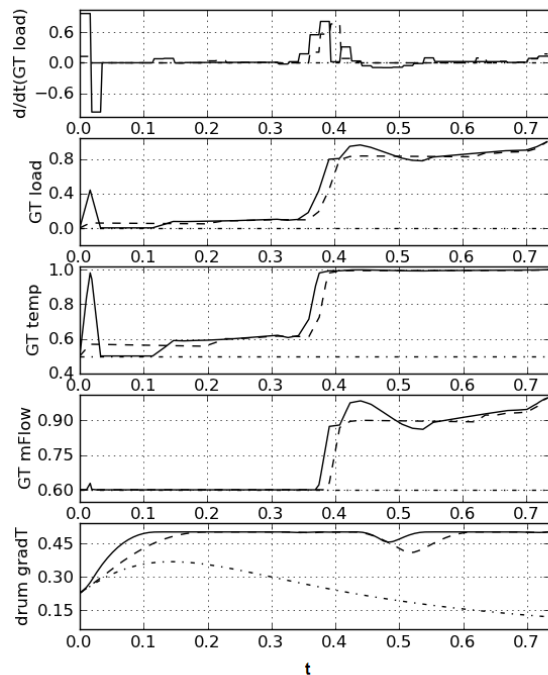


Figure 4: Optimal start-up trajectories for 1 DOF: dashed (monotonically increasing power output) and solid (non-monotonic power output) curves. Simulated initial guess trajectories are shown by dash-dotted curves. From the top: the derivative of the GT load, the GT load, the GT outlet temperature, the GT mass flow and the temperature gradient in the wall of the drum. All results and times have been normalized.

The optimal and normalized time for the GT to reach 95% of full load is approximately the same in both cases: 0.724 and 0.723 for the monotonic and non-monotonic load profile, respectively. From Figure 4, it can be seen that the temperature gradient constraint becomes rapidly active in spite of the low GT load. This is due to that the pressure controller keeps the bypass valve closed which results in a low mass flow through the valve and a high pressure in the

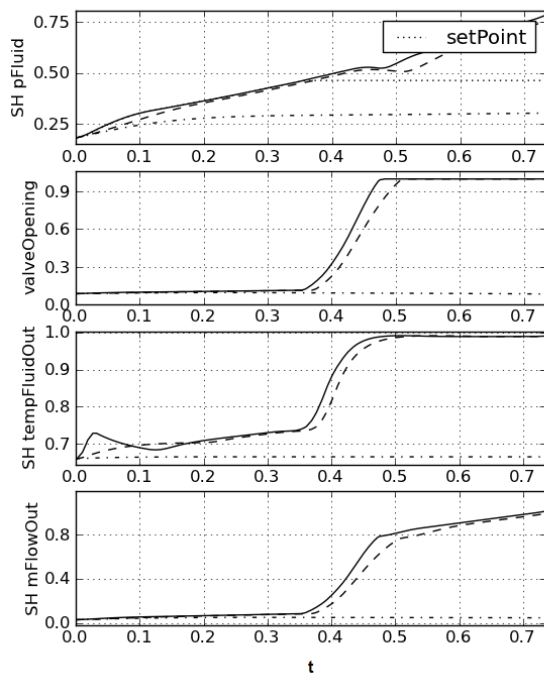


Figure 5: Optimal start-up trajectories for 1 DOF: dashed (monotonically increasing power output) and solid (non-monotonic power output) curves. Simulated initial guess trajectories are shown by dash-dotted curves. From the top: the pressure in the superheater on the steam/water side and the pressure control loop set-point, the bypass valve opening in the pressure controller, the outlet temperature and the outlet mass flow from the superheater on the steam/water side. All results and times have been normalized.

HRS, giving large temperature gradients in the wall of the drum. There is also hot steam in the HRS due to transients from phase 1 and from the fact that the start-up is considered as a hot start. At about $t = 0.37$, the GT load is rapidly increased from about 10% to 80%, at an optimal rate that steadily maintains the gradient constraint active. At about $t = 0.44$, the non-monotonic load profile reaches a maximum of 90% before decreasing to 80% at $t = 0.51$. This behavior is related to the optimization formulation that penalizes deviations from the reference load of 100% and may therefore lead to overshoots before the gradient constraint becomes too constraining. The overshoot that is allowed at low input penalty coefficient β is not observed in the case of a monotonically increasing load. The dip in the temperature gradient observed at about $t = 0.48$ is due to the limited degree of freedom and its amplitude decreases with an increasing discretization

level. In the case of a monotonically increasing load, the gradient dip cannot be avoided and is rather independent on the discretization level. After scaling the value of the objective function is 1 for the monotonically increasing power output case and 0.95 in the non monotonic case.

4.3 2 DOF Optimization of CCPP2

The CCPP2 model contains 28 continuous time states and 389 scalar equations. When optimizing the 2 DOF case the input signal representing the power output was defined as non monotonic. The second input, the opening of the bypass valve, could vary from closed to fully open with a derivative in the interval $[-0.5, 0.5]$. The model has been successfully optimized to full load, see results in Figure 6 where the solid trajectory represents the solution of the optimization problem.

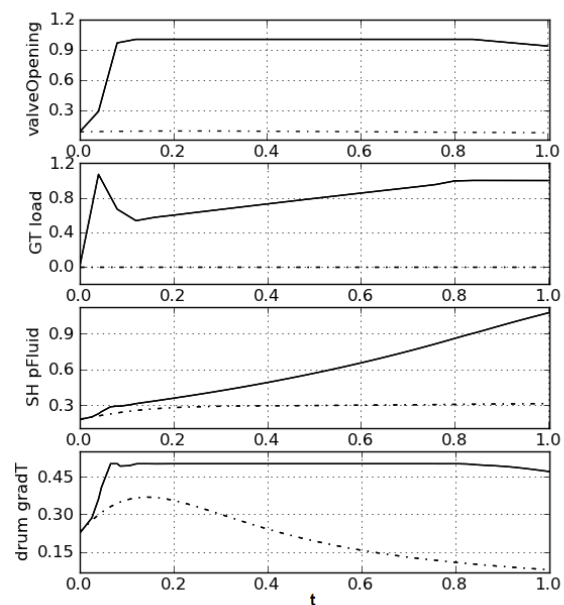


Figure 6: Optimal start-up trajectories for 2 DOF: solid (non-monotonic power output) curve. Simulated initial guess trajectories are shown by dash-dotted curves. From top: the bypass valve opening, the power output, the pressure in the superheater on the steam/water side and the temperature gradient in the wall of the drum. All results and times have been normalized.

The optimal time for the GT to reach 95% of full load is approximately $t = 0.75$ when the model with 2 DOF is optimized, see Figure 6. The temperature gradient constraint is active from $t = 0.06$ and the GT load

can not increase as rapidly as initiated after $t = 0.04$. At about $t = 0.12$ the GT load increases steadily at almost constant rate to not violate the temperature gradient constraint until it reaches its maximum value at $t = 0.84$. The dip in the temperature gradient that was observed in the 1 DOF case is not observed. The gradient constraint is not completely active around $t = 0.1$ which most likely is due to the discretization. The bypass valve is opened at $t = 0$ and is fully opened at $t = 0.12$, inducing that the power load can be increased more rapidly for $t < 0.5$, comparing to the 1 DOF case. After scaling the value of the objective function is 0.19 in the 2 DOF non monotonic case, and this value also includes a contribution from the dv/dt term in the cost function.

The total power produced during the start-up procedure corresponds to the area under the graph of the power output. Even though the GT reaches full load later than in the 1 DOF case, the 2 DOF model produces more GT power during the start-up than the 1 DOF model. The objective function value corresponding to the deviation of the power output from full load is thus about 1/5 of the 1 DOF model solution. This result shows the benefit of using an extra degree of freedom.

4.4 2 DOF Optimization of CCPP3

The model CCPP3 contains 39 continuous time states and 576 scalar equations. Two different optimization problems based on CCPP3 are considered in this section.

4.4.1 Constant Temperature Gradient Bounds

An optimization problem based on CCPP3 with constant bounds on temperature gradients has been successfully solved, where full load is reached, see the results in Figures 7 and 8, dashed curves.

The degrees of freedom were du/dt and dv/dt . The GT load input u was non-monotonic and the bypass valve was controlled in the optimization so that the opening of the bypass valve could vary from closed to fully open with a derivative in the interval $[-0.5, 0.5]$. The optimal time for the GT to reach 95% of full load was approximately $t = 0.45$, see Figure 7.

The GT load can not increase as rapidly as initiated after $t = 0.04$ since at the end time of the second block ($0.04 < t \leq 0.08$, since the degree of freedom du/dt is piecewise constant) the header constraint is active. The header temperature gradient constraint is active from $t = 0.08$ until $t = 0.3$. The drum tempera-

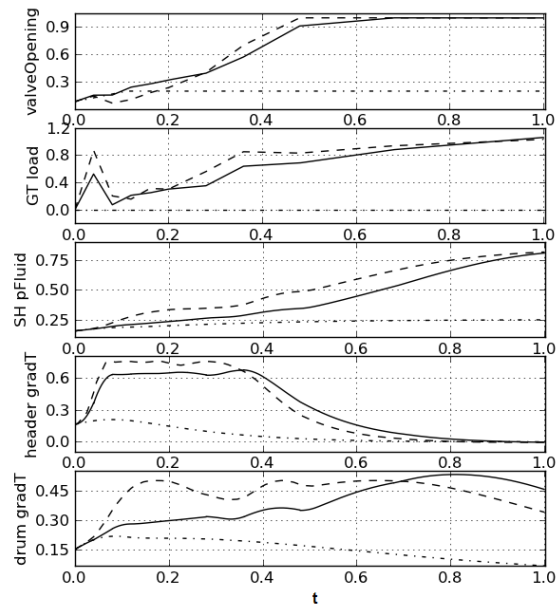


Figure 7: Optimal start-up trajectories of CCPP3. Dashed curves show results for constant temperature gradient bounds and solid curves show results for pressure dependent constraints. Simulated initial guess trajectories are shown in dash-dotted curves. From the top: bypass valve opening, GT power output, pressure in the superheater on the steam/water side, temperature gradient in the wall of the header and temperature gradient in the wall of the drum. All results and times have been normalized.

ture gradient constraint is active at different times from $t = 0.16$ and it is the only active temperature gradient constraint when $t > 0.3$. Around $t = 0.6$ the drum temperature gradient constraint is active for the longest time sequence.

From $t = 0.12$ the GT load increases with a rate that varies to not violate the header drum constraint. After $t = 0.2$ the GT load increases with a steady almost constant rate until it reaches about 80% of full load at $t = 0.34$. The drum temperature gradient constraint is not active during this time period. From $t = 0.34$ the GT load increases at a low rate to not violate the drum temperature gradient constraint until it reaches its maximum value of 1 at $t = 0.88$. The bypass valve is opened at $t = 0$ and is fully opened at $t = 0.48$. The valve though closes at $t = 0.08$ giving a rise in the HRSG pressure and the drum temperature gradient.

After scaling the value of the objective function is 0.34, and this value also includes a contribution from the dv/dt term in the cost function.

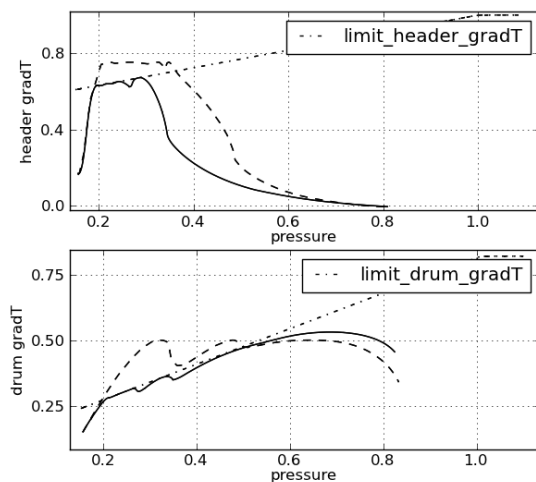


Figure 8: Optimal start-up trajectories of CCPP3. Dashed curves show results for constant temperature gradient bounds and solid curves show results for pressure dependent constraints. The dash-dotted curve shows the pressure dependent stress constraint. From the top: temperature gradient in the wall of the header as a function of pressure and the temperature gradient in the wall of the drum as a function of pressure. All results and times have been normalized.

4.4.2 Pressure Dependent Temperature Gradient Constraint

In Figures 7 and 8, the results for the case when applying pressure dependent temperature gradient constraints to CCPP3 are shown in solid curves.

The GT load input u was non-monotonic and the bypass valve was controlled in the optimization so that the opening of the bypass valve could vary from closed to fully open with a derivative in the interval $[-0.5, 0.5]$. The degrees of freedom were du/dt and dv/dt .

The optimal time for the GT to reach 95% of full load was approximately $t = 0.56$. As in the case of constant stress bounds, the GT load can not increase as rapidly as initially after $t = 0.04$, since at the end time of the second block ($0.04 < t \leq 0.08$, since the degree of freedom du/dt is piecewise constant) the header constraint is active, see Figures 7 and 8.

The header temperature gradient constraint is active from pressures $p=0.19$ to $p=0.25$ corresponding to the time period $t = 0.07$ and until $t = 0.24$. The drum temperature gradient constraint is active from pressures $p = 0.2$ to $p = 0.55$ corresponding to the times $t = 0.09$ to $t = 0.7$. The GT load is thus more constrained at

lower pressures than in the previous case, see Figure 8.

The pressure dependent stress constraints allow the drum temperature gradient to attain larger values when $t > 0.7$ (the pressure in the HRSG is larger than 0.55) compared to the constant constraint used in the previous case. The stress in the header is, however, more constrained in second case, which yields a lower rate of increase of the GT load as compared to the previous case. From $t = 0.08$ the GT load increases with a rate that does not violate the header drum constraints. The bypass valve is opened at $t = 0$ and is fully opened at $t = 0.65$. The pressure is kept at low values when the temperature gradient constraints are active and the pressure can increase at a higher rate when $t > 0.5$.

After scaling the value of the objective function is 0.47 and this value also includes a contribution from the dv/dt term in the cost function.

4.5 Discussion

When starting up a power plant, the most desirable goal does not necessarily have to be to reach full load as fast as possible. To achieve as much power output as possible during the start-up procedure could be just as important. The results show how a larger amount of produced power during start-up can be achieved when adding the opening of the bypass valve as degree of freedom.

When using the pressure controller in the 1 DOF model, it has been shown to function in a far from optimal way since the pressure is controlled so that the load cannot increase during the first 0.35 s. The set point of the controller could be modified so that the bypass valve can be opened earlier in the start-up, lowering the pressure in the HRSG and giving the load more operational space where it does not violate the gradient constraint. When the bypass valve opening is used as a degree of freedom in the 2 DOF case the valve is opened earlier, lowering the pressure in the HRSG and allowing the load to be increased earlier. The results from the 2 DOF case produces the most power during the start-up and the benefits from using two degrees of freedom instead of one is clear.

The 2 DOF model produces more steam in an earlier phase of the start-up due to the faster ramp up of the GT load. It is though not taken into account in this paper to determine if the produced steam is of sufficient quality to start the third phase of the start-up procedure; the loading of the ST. A more complete picture of the efficiency of the start-up could be attained by modifying the objective function and adding

more complex and thorough descriptions of possible objectives, so that the efficiency is maximized and the economical costs during the whole start-up transient is minimized. Thus the economical aspects of not only the load produced could be taken into account.

The rate of increase of the GT power load has been unlimited in all optimizations done in this paper. This and the fact that the GT power load is allowed to decrease gives a peak in the GT power load in the beginning of the start-up phase for all non-monotonic GT load cases. From an optimization point of view this is a satisfying result, since it is clear that the optimizer is trying to make the load reach its reference value as fast as possible. The GT load must though decrease to not violate the temperature gradient constraints. In actual power plants, such a fast increase in load could damage the GT or even not be physically applicable. By penalizing the use of the du/dt input such peaks could be avoided in the optimization.

When comparing the two optimization problems based on CCPP3, presented in Sections 4.4.1 and 4.4.1 respectively, it is clear that the time to reach full load is more or less the same even though in the second case there are stricter temperature gradient constraints at lower pressures comparing to the first case. The optimizer compensates this by using the full potential of the drum gradient constraint for $t > 0.7$. The rate of increase of the GT load is slightly larger for second case when $t > 0.7$ and it can be observed that the pressure increases at a higher rate than in first case. Even though the time to reach full load is approximately the same the profile resulting in the second case keeps the lifetime consumptions of the stressed components at a level used in actual power plant controls. The header constraint is active earlier in the start-up phase comparing to the drum constraint. This is due to that the hot exhaust gas from the GT enters the header first when the gas is of the highest temperature. The exhaust gas reaches the drum with a time delay and the exhaust gas is of lower temperature than when reaching the header. The drum consists of water in its fluid state and the gradient is therefore coupled to the pressure in the component. Steam is though decoupled from pressure and the header temperature gradient is more dependent on the GT exhaust gas temperature than the pressure. The basic stress model used in the second case uses constraints that are typically used in power plant control. Since the stress levels obtained with constant temperature gradient bounds violate these constraints, the result from the second case is the most preferable choice.

For additional background, results and discussions from this project, see [17].

5 Summary and Conclusions

In this paper it has been shown how a start-up procedure of a combined cycle power plant can be optimized with respect to the start-up time and the power production during start-up, using JModelica.org. The thermal stress in the heat recovery steam generator has been considered as the most limiting constraint when starting up the GT to full load, i.e. its maximum power output.

Three different optimization models have been considered; one where the load u of the gas turbine is the sole degree of freedom and two where both the load u and the opening v of the bypass valve are degrees of freedom. Also, two different levels of model fidelities have been considered. Based on these, four optimization problems have been successfully solved where the power output has been controlled to the reference value of 100% and it has been observed that by adding the opening of the bypass valve as degree of freedom a larger amount of power during start-up is produced. In addition, it has been shown how pressure dependent stress constrains contributes to increased lifetime of critical components, which maintaining fast start-ups.

The models have been adapted to suit optimization purposes concerning the start-up of the GT and thus the ST has not been modeled. The next step towards achieving more realistic results could be to close the steam cycle and to include more detailed components in the model. More constraints could as well be used and additional degrees of freedom could be added. It has not been taken into consideration when it is most optimal to start the ST and if the optimization of the GT loading should take this factor into account. One improvement could thus be to find when, during the start-up procedure, the ST should be started and to determine when and how much of the steam should pass the bypass valve. Another improvement could be to include economical aspects and to minimize the fuel spent during start-up while maximizing the produced power load. The work presented in this paper is one step towards an optimal power plant control and could be used with an on-line strategy such as model predictive control.

Acknowledgments

The German Ministry BMBF has partially funded this work (BMBF Förderkennzeichen: 01IS09029C) within the ITEA2 project OPENPROD (<http://www.openprod.org>). Modelon's contribution to this work was partially funded by Vinnova within the ITEA2 project OPENPROD (dnr: 2010-00068). Johan Åkesson acknowledges financial support from Lund Center for Control of Complex systems, funded by the Swedish research council.

References

- [1] Casella, F. and Pretolani, F. Fast Start-up of a Combined-Cycle Power Plant: A Simulation Study with Modelica. In: Modelica Conference, pp. 3-10, Vienna, Austria, 2006.
- [2] Casella, F., and Leva, A. Modelica open library for power plant simulation: design and experimental validation. In: Proceedings of 3rd International Modelica Conference, pp. 41-50. Linköping, Sweden, 2003.
- [3] Casella, F., Farina, M., Righetti, F., Scattolini, R., Faille, D., Davelaar, F., Tica, A., Gueguen, H. and Dumur, D. An optimization procedure of the start-up of combined cycle power plants. In: 18th IFAC World Congress, pp. 7043-7048. Milano, Italy, 2011.
- [4] Casella, F., Donida, F. and Åkesson, J. Object-oriented modeling and optimal control: a case study in power plant start-up. In: 18th IFAC World Congress, pp. 9549-9554. Milano, Italy, 2011.
- [5] Shirakawa, M., Nakamoto, M. and Hosaka, S. Dynamic simulation and optimization of start-up processes in combined cycle power plants. In: JSME International Journal, vol. 48 (1), pp. 122-128, 2005.
- [6] Dassault Systemes. Dymola, <http://www.3ds.com/products/catia/portfolio/dymola>, 2012, viewed 2012-06-12.
- [7] Kehlhofer, R., Warner, J., Nielsen, H., Bachmann, R. Combined-Cycle Gas and Steam Turbine Power Plants, second edition, PennWell Publishing Company, 1999.
- [8] Biegler, L., Cervantes, A., Wachter, A. Advances in simultaneous strategies for dynamic optimization, Chemical Engineering Science 57, pp. 575-593, 2002.
- [9] Modelon AB. JModelica Home Page. <http://www.jmodelica.org>, 2009, viewed 2012-06-12.
- [10] Andersson, C., Åkesson, J., Führer, C., Gäfvert, M. Import and Export of Functional Mock-up Units in JModelica.org. In: 8th International Modelica Conference 2011. Modelica Association, 2011.
- [11] Åkesson, J., Årzen, K.E., Gäfvert, M., Bergdahl, T., and Tummescheit, H. Modeling and optimization with Optimica and JModelica.org - languages and tools for solving large-scale dynamic optimization problems. In: Computers and Chemical Engineering, vol. 34 (11), pp. 1737-1749, 2010.
- [12] Biegler, L. Nonlinear Programming: Concepts, Algorithms, and Applications to Chemical Processes, SIAM, 2010.
- [13] Biegler, L., Wächter, A. On the Implementation of a Primal-Dual Interior Point Filter Line Search Algorithm for Large-Scale Nonlinear Programming, Mathematical Programming 106(1), pp. 25-57, 2006.
- [14] Åkesson J. Languages and Tools for Optimization of Large-Scale Systems, PhD Thesis ISRN LUTFD2/TFRT-1081-SE, Regler, 2007.
- [15] Modelica Association. The Modelica Association, <https://www.modelica.org>, 2012, viewed 2012-06-12.
- [16] Bauer, O. Modelling of Two-Phase Flows with Modelica, Master's Thesis, Lund University, Department of Automatic Control, 1999.
- [17] Lind, A., Sällberg, E. Optimization of the Start-up Procedure of a Combined Cycle Power Plant, Master's Thesis, Lund University, Department of Automatic Control, 2012.

Modeling and Simulation of a Vertical Wind Power Plant in Dymola/Modelica

Joel Petersson*

Joel.petersson@gmail.com

Hubertus Tummescheit†

hubertus.tummescheit@modelon.com

* Lund University

Sölvegatan 18

SE-22100, Lund, Sweden

www.lth.se

Pär Isaksson*

Par.Isaksson.lth@gmail.com

Johan Ylikiiskilä†

johan.ylikiiskila@modelon.com

† Modelon AB

Ideon Science Park

SE-22370, Lund, Sweden

www.modelon.com

Abstract

A small wind power plant connected to the grid has been modeled in Modelica/Dymola and controlled using external controllers written in C++. The small wind power plant consists of three wind power units, with a nominal power of 3kW, and one grid connection interconnected with an internal DC-grid. All the controls needed for controlling and optimizing the operation of the individual parts in the plant were developed and implemented. Apart from this a managing control for the entire plant were developed and implemented.

The control was implemented using an external static library interconnected with Dymola. the External Object approach for implementing objects in Modelica was also tested. The optimization algorithms developed for the wind turbine was done in a way so that no measurements of the wind speed are needed. The controls were developed so that they can achieve a number of different tasks such as Reactive Power Compensation and Island Control.

Models were implemented in Modelica using Dymola as tool. In order to model the power electronics involved in the system the Electric Power library (EPL) has been utilized. Models for the wind turbine were developed and tested.

The models were in the end tested and evaluated by running a number of different simulations. The Different test cases consists of optimizing the power output, controlling the power output to a desired level and island operation, that is to power up a small grid on its own.

Keywords: wind power, power electronics, control, optimization, vertical wind power, Electrical Power library

Nomenclature

P	Power, if electrical active Power.
C_p	Efficiency coefficient of the wind turbine.
T_ω	Mechanical torque.
β	Pitch angle of the rotor blades.
λ	Tip speed ration of the rotor blades.
λ_i	Factor used for calculating C_p .
ω_T	Rotational speed of the turbine.
ρ	Air density.
c_{1-6}	System dependent constant used to calculate C_p .
A	Area swept by the rotor.
R	Radius of the wind turbine.
v	Wind speed.
v_{base}	Wind model base component.
v_{gust}	Wind model gust component.
v_{noise}	Wind model noise component.
$L_{sd/q}$	Inductance in d/q-axes.
T^{ref}	Torque reference.
Ψ_m	Permanent flux.
$i_{sd/q}^{ref}$	Direct/Quadrature current reference.
$i_{sd/q}$	Direct/Quadrature current.
T	Electrical torque.
pp	Number of pole pairs in the generator.

1 Introduction

Wind power is at the moment in a globally expansive phase with different kinds of technical solutions and suppliers. In most solutions power electronics is incorporated to smaller or larger extent. Most wind power plants currently operate with a horizontal axis turbine, however vertical axes turbines is an interesting future

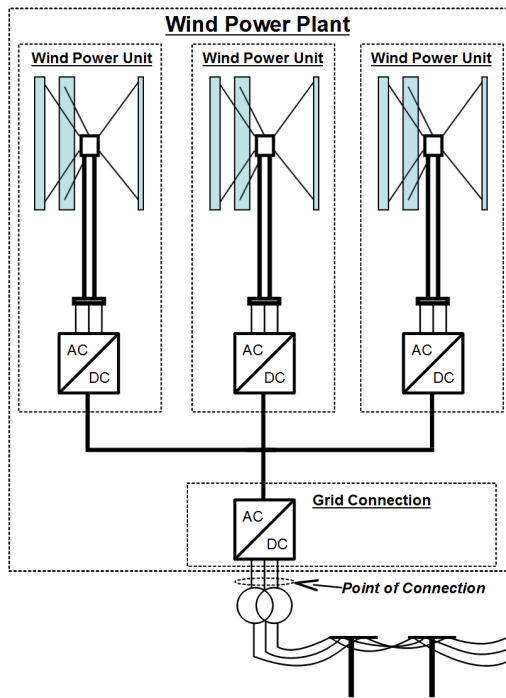


Figure 1: Schematic picture of how the wind power system is set up.

alternative. Vertical plants have been erected and are now being tested in Sweden.

The advantages of using a vertical axes turbine are a simple and robust construction with a minimal amount of moving parts, which allows for a cost efficient wind power plant with an aspect both to investment, operation and maintenance. Other pros are independence of wind direction, less sensitivity to turbulence, simple blade profiles and lower noise levels.

The objective of this project was to model and control a small wind power plant consisting of one or several wind power units and one grid connection interconnected with an internal DC-grid. The configuration of investigated can be seen in figure 1. This article will mainly focus on the case with one single wind power unit connected to the grid through back to back full inverters¹. The models and controls developed will be tested by running different test cases. They will also be evaluated according grid codes, see [1] & [2].

2 Modeling

The system to be modeled consists, as depicted in figure 2, of a wind turbine connected through a break to

¹Back to back full inverters consists of two inverters coupled by a DC-grid, i.e. the DC-connection of the generator inverter is coupled to the DC-connection of the grid inverter.

a generator. The power voltage output from the generator is rectified by an inverter connected to a DC-grid which is then connected via another inverter to a three-phase grid. The reason to why an internal DC-grid is utilized is mainly to decouple the rotational speed of the wind power units from the grid frequency, the DC-grid also acts as both filter and buffer.

2.1 Wind Turbine

Wind turbine power generations depend on the interactions between the wind and the rotor. The power extracted from the wind by the rotor can be described as the kinetic energy of the wind times an efficiency coefficient. The efficiency coefficient is varying with the pitch angle of the blades and tip speed ratio. The tip speed ratio is the wind speed relative the speed of the tip of the turbine's blades. The mechanical power P in a vertical wind power unit can be described by equation 1. The efficiency coefficient, C_p , can be described according to equation 2, as proposed in [3] and [4].

$$P = \frac{1}{2} \cdot \rho \cdot A \cdot v^3 \cdot C_p \quad (1)$$

$$C_p(\lambda, \beta) = c_1 \cdot \left(\frac{c_2}{\lambda_i} - c_3 \cdot \beta - c_4 \right) \cdot e^{-\frac{c_5}{\lambda_i}} + c_6 \cdot \lambda \quad (2)$$

$$\frac{1}{\lambda_i} = \frac{1}{\lambda + 0.08 \cdot \beta} - \frac{0.035}{\beta^3 + 1} \quad (3)$$

$$\lambda = \frac{\omega_T \cdot R}{v} \quad (4)$$

The area swept by a vertical wind power turbine is simply expressed as the rotor diameter times the rotor length. The efficiency coefficient, C_p , can be calculated according to equation 2 to 4. A typical C_p -curve for different pitch angles² can be seen in figure 3. The mechanical torque T_ω can be obtained by dividing the power absorbed, P , with the rotational speed of the turbine, ω_T . A component modeling the effect of tower shadow was also implemented according to [5]. The tower shadowing effect occurs when a rotor blade passes behind the tower, this since it is then shadowed from the wind by the tower. This was modeled by subtracting a torque component each time a rotor blade passes behind the tower.

$$T_\omega = \frac{P}{\omega_T} \quad (5)$$

²The pitch angle is the angle at which the rotor's blade surface contacts the wind.

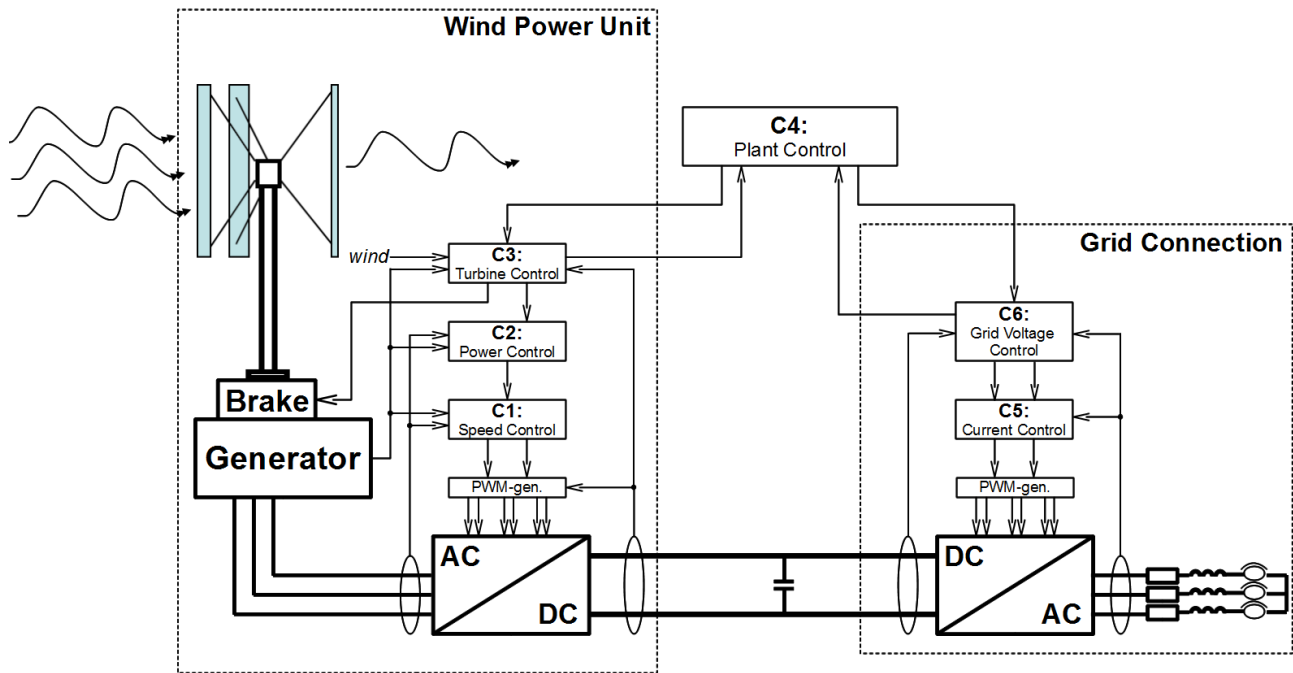


Figure 2: Overview of control structure and configuration of the wind power plant.

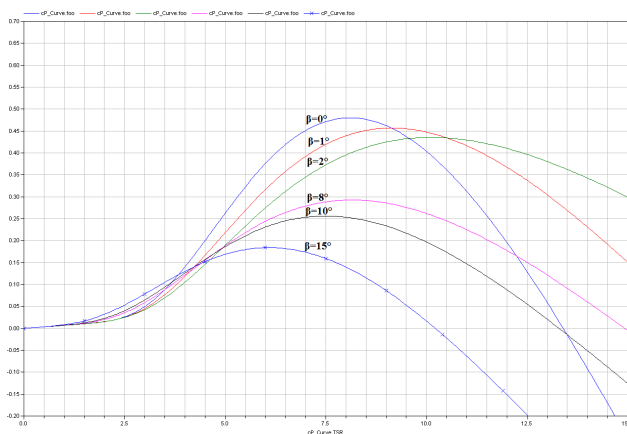


Figure 3: Typical characteristic of a C_p -curve at different pitch-angles, β

Apart from modeling the wind turbine as an energy producing unit modeling of the actual mechanical and electrical properties also needs to be done in order to get a good overall model. The wind turbine was modeled to be directly coupled to a shaft and via a brake to the permanent magnet generator. The shaft was modeled as inertia, containing both the rotor's inertia and the actual shaft's inertia, coupled via a model of coulomb friction in bearings, in order to simulate losses in the shaft, to the brake. For modeling of the inertia, bearing friction and brake components from Modelica's standard library was used.

2.2 Wind Model

The produced power of a wind turbine is tightly linked to the current wind speed. The wind changes both during the day and the seasons. In order capture these changes and to simulate the real wind conditions, a wind model consisting of three components is used in this project. The three components are:

- A base component, v_{base}
- A gust component, v_{gust}
- A noise component, v_{noise}

The three components are summarized to $v = v_{base} + v_{gust} + v_{noise}$. The base component is always present, it may be constant, a ramp signal or have any other form. The gust component appears randomly during time and the noise component is modeled as white noise.

2.3 Power Electronics

The system is designed using back to back full inverters. This means that the modeling of the power electronics becomes essential for the complete model. This since the power electronics is used for controlling both the individual wind power units and grid connection. The Electric Power Library was chosen to be the main tool used for modeling the power electronics and generators since it is well suited for the task.

2.3.1 The Electric Power Library

The Electric Power library is a Modelica library used for modeling of power electronics and can be used in both steady state and transient mode for the simulations and initializations. The Electric Power library provides components for modeling AC three phase system, AC one phase systems and DC system. The AC three phase systems can be represented in the abc-, dq0- and dq-frame. Especially modeling in the dq0-/dq- reference frame provides relatively quick simulations, compared to simulations in the abc-reference frame. This since a symmetrical three phase voltage or current is represented by constants in the dq0-reference frame. The Electric Power library was originally written by H.J. Wiesmann and is currently owned and sold by Modelon AB.

The Electric Power Library supports modeling in the dq0-reference frame. The dq0-reference frame not only simplifies analysis of the system but also increases the simulation speed.

The main components used from the Electric Power library were models for the inverters, a model for a Permanent Magnetized Synchronous Machine, PMSM, DC-link model and models for transmission lines. The rotor connection of the PMSM model is compatible with the Modelica standard library.

3 Control Design

In order to control the wind power plant different controls are needed. The controls are developed to achieve a number of different operation modes that are needed in order to achieve the grid codes. The overall control structure developed can be seen in figure 2. A short description of the different controllers follows below, for more detailed descriptions see [6].

3.1 Wind Power Unit Control

The control needed to control the wind power unit has been divided into three controllers the Turbine Control, the Power Controller and the Speed Controller.

3.1.1 Turbine Control

The Turbine Control's task is to manage the wind power unit, which is to decide when the unit should start and stop as well as to give instructions as to which mode the unit currently should be working in. The Turbine Control should communicate with both the

Plant Control and the Power Controller. The communication should be kept to a minimum and no actual control should be done by the Turbine Control and Plant Control. The controller's main task is to make decisions about when the unit could be in operation and provide information about the unit's current capacity to the Plant Control. In order to do this the Turbine Control needs information about the current wind speed as well as orders from the Plant Control. Apart from this the control also needs information about how fast the rotor is spinning. This to avoid using the brake at high speeds and instead do a soft deceleration using the generator.

3.1.2 Power Controller

In order to control the power output from the wind power unit the Power Controller was developed. By controlling the power output from the wind power unit a number of different control modes can be achieved. The Power Controller is designed in two different sections.

The first part's task is to set an appropriate power reference for the second part. This power reference mainly depends on what control mode that is desired. For example when ordered to control the DC-voltage level the Power Controller receives a power offset. The power offset received is the current power input needed from the wind power unit in order for the total power input to be equal to the power output, thus keeping the DC-voltage level inert. The Power Controller then calculates the power reference as a deviation from the power offset in order to control the DC-voltage level.

The second part's task is to control the power output from the wind power unit to the reference value. This is done by adjusting the rotational speed of the turbine. As long as the wind speed is high enough for achieving the desired power the task is quite trivial and easily achieved by a PI-controller. However when the wind speed is too low and the desired power cannot be reached the controller should do as good as possible. In this case the controller should maximize the power output from the plant. The algorithm used to optimize the rotational speed was based on the sensorless maximum power point tracking algorithm proposed in [7].

A flow chart over the general operation of the Power Controller can be seen in figure 4. The general idea with the control is to determine whether the current operation point is to the left or right of the optimal point, and depending on this take different actions.

The control algorithm starts by setting initial con-

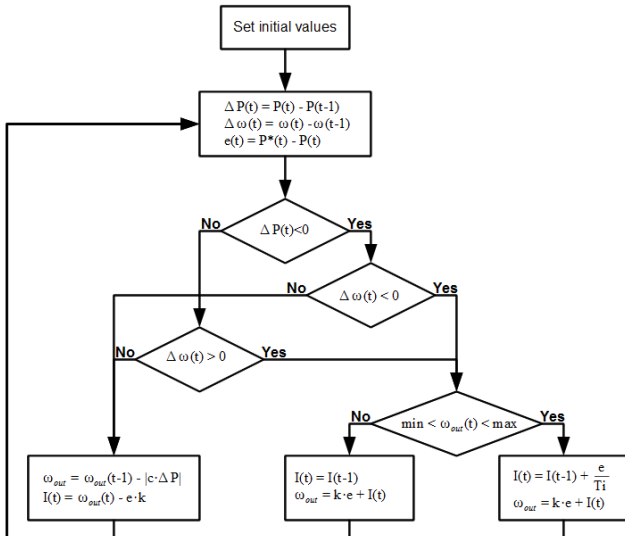


Figure 4: Flow chart for the power control algorithm

dition and reads new measurements. It then tries to decide whether the current operation point is to the left or right of the optimal by comparing the change in power output and the change in rotational speed. If the current operating point is considered to be on the left side of the optimal point the control output is calculated by a PI-controller with a simple anti-windup. If it is considered to be on the right side of the optimal point the control output is calculated as $\omega_{out}(t) = \omega_{out}(t-1) - |c \cdot \Delta P|$ where c is a control constant and ΔP is the change in power. Also the integral for the PI-controller is updated in order to achieve a bumpless transfer between the two controls. That is when a control switch is made the output from the “new” control is equal to that of the “old”.

3.1.3 Speed Controller

The speed controller consists of a series of cascaded PI- and PIE-controllers. The PIE-controller is a PI-control which is compensated by feed-forwarding the back electromagnetic force from the generator. The control parameters are calculated based on the generator parameters. The outer controller is a PI-controller controlling the rotational speed of the turbine by generating a torque reference. The requested torque can be achieved in many different ways, according to equation 6, where T , pp , ψ_m , L_{sd} and L_{sq} are generator parameters and i_{sd} and i_{sq} are currents in the respective axes. In this project a method was chosen where the direct current, i_{sd} , is set to zero, which generates the current references in equation 7.

$$\frac{T}{pp} = \psi_m i_{sq} + (L_{sd} - L_{sq}) i_{sd} i_{sq} \quad (6)$$

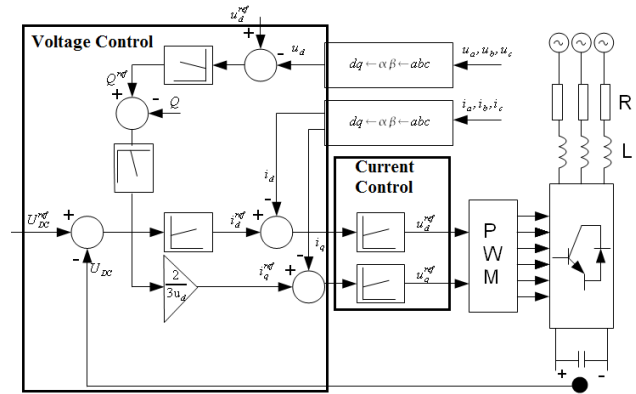


Figure 5: Schematic picture of the control structure used for controlling the grid connection.

$$\Rightarrow \begin{cases} i_{sd}^{ref} = 0 \\ i_{sq}^{ref} = \frac{T^{ref}}{\psi_m \cdot pp} \end{cases} \quad (7)$$

The two inner PI-controllers control the currents of the PMSM which are designed according to [8].

3.2 Grid Connection Control

The control needed for the grid connection has been divided into two controllers the Voltage Control and the Current Control.

3.2.1 Voltage Control

The Voltage Control’s task is to control the active and reactive power flow at the point of connection. This can be done in different ways depending on which control goal is desired. For example Swedish wind power plants are supposed to deliver zero reactive power. However in a small system the reactive power flow can be controlled so that the output voltage level to the grid is kept constant, independent of the active power output. Another possible control mode is Island operation in which the plant powers up the grid and controls the grid’s voltage and frequency.

3.2.2 Current Control

The current controllers are implemented as two parallel PI-controllers controlling the direct and quadrature currents. The current references are received from the voltage control. The currents are controlled by outputting a reference voltage to the transistor control, modulator. The modulator uses the requested voltages

in order to control the transistor by generating by generating switch signals for each of the individual inverters.

3.3 Wind Power Plant Control

The Plant Control's task is to manage the wind power plant. Most of its operation consists of setting the control modes of the grid connection and wind power units to achieve a specific control goal. The control could be done very simplistic, very advanced or anyway in between. The implementation done here was kept quite simplistic with some intelligence, for example loss compensation and ability to choose the number of plants that should be in operation and is covered in [6].

4 Implementation

4.1 Wind Power Unit

Most of the implementation was done using Modelica's standard library and Modelon's Electric Power library, however some models for the wind turbine was developed as well as a wind model.

An overview of the Modelica model over a wind power unit can be seen in figure 6, to the right is the top view, with inverter and controllers, and to the left is the contents of the actual unit, with generator, shaft and turbine model. The unit model consists of a model of the turbine connected to a shaft, modeled as an inertia and a bearing friction. The shaft is connected via a brake to the permanent magnet synchronous generator, PMSG, the electrical output from the PMSG is connected to an inverter which performs an AC to DC conversion. Additionally a wind model was developed in order to model the wind in a realistic way. The inverter is controlled by the control-blocks on top of it, and by performing the AC/DC conversion the inverter controls the generator. As can be seen most models incorporated in the wind power unit model are from either Modelica's standard library or Modelon's Electric Power library. The models implemented in this project are the turbine model, wind model, the controllers and an interface between the control and the inverter. The model of the turbine was implemented using equations 1 to 5.

4.2 Grid Connection

The grid connection is modeled using components from the Electric Power library. The grid model con-

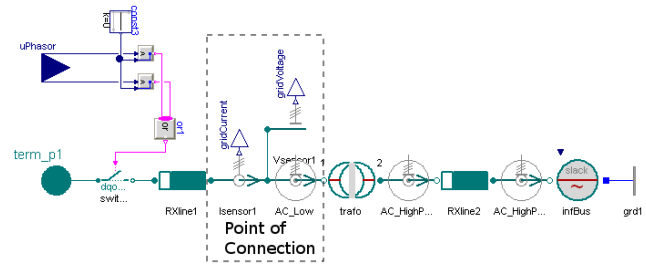


Figure 7: Screenshot of a strong grid model implemented in Dymola.

sist of a transmission line to the point of connection. After the point of connection follows a transformer and another, longer, transmission line. The model shown in figure 7 is depicting a model of a strong grid. In order to model a strong grid an optimal voltage source is connected to the long transmission line. The switch implemented before the first transmission line is necessary since the average inverter model used is based on a voltage source which means that when in passive mode it acts as a ground connection.

4.3 Control

There are two possibilities when trying to implement C/C++ objects into Dymola/Modelica. The first is using an External Static Library and the second is using the External Object function in Modelica. Both methods have their pros and cons. The External Static Library is disconnected from the modeling tool to a greater extent while the External Object is more interconnected with Modelica. In the end the decision was made to mainly use the External Static Library for implementation of the control structure. This was due to two different reasons.

1. The goal was to be able to run the exact same code both in simulations and on the actual plant and for this reason it was desirable to separate the control code and the models as much as possible.
2. The External Object currently only supports code in FORTRAN 77 and C [9], while the static external library supports both C and C++.

However the External Object was also investigated and tested. The control blocks implemented in Dymola can be seen in figure 6 and 8 while a detailed description is covered in [6].

4.4 Wind Power Plant

Two different Modelica models were developed, however the only difference between the two is the num-

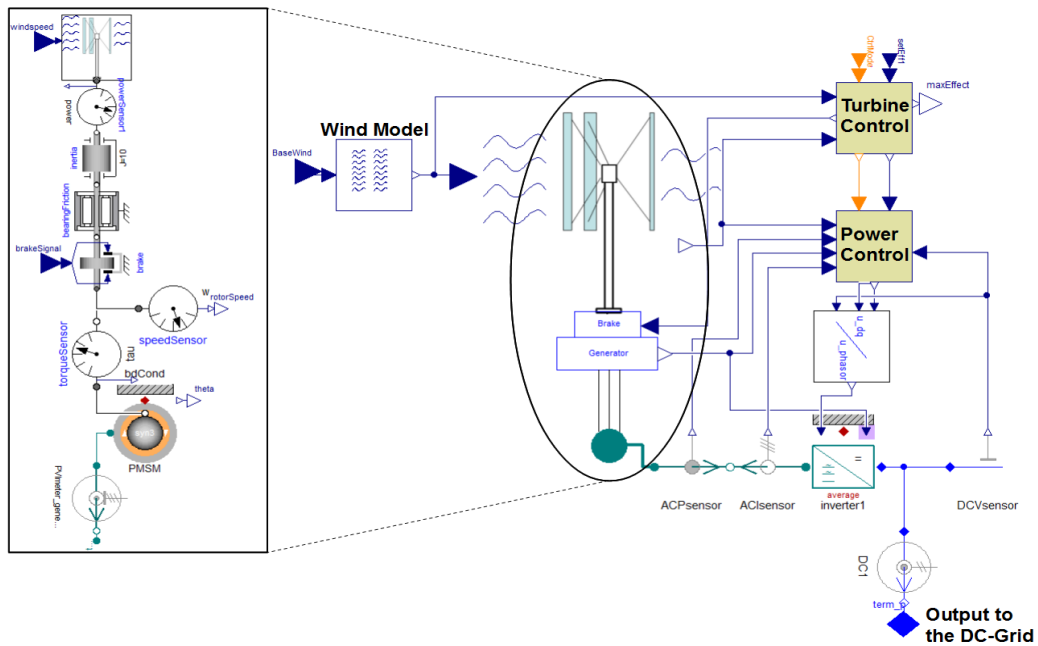


Figure 6: Screenshot of the wind power unit model in Dymola.

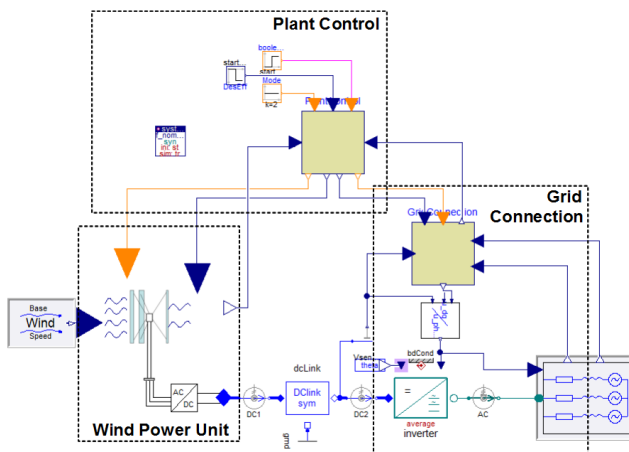


Figure 8: Screenshot of a wind power plant model using one wind power unit in Dymola.

ber of wind power units that is connected to the DC-grid. The models were constructed by connecting one or more wind power unit models to a model of a DC-link, the DC-link was then connected to the grid connection model. In reality the wind acting on the different wind power units is not identical. The wind model was moved to the wind power unit model to reflect this fact.

5 Results

5.1 Optimizing Operation Point

This test is designed for evaluating performance of the optimizing algorithms of the Power Controller. The task in this case is to maximize the power output from the unit when the wind speed is not high enough to achieve the requested power output. The unit starts at standstill.

The results from the simulation are shown in figure 9. As can be seen the wind speed starts at 8 m/s and after 200s it increases to 11 m/s over 100s. When observing the power output from the unit it can be noticed that the output is almost zero until the rotational speed reference has been reached, this since the generator is not applying any negative torque. After this the power output is approximately 1 kW, and when observing the C_p -value in figure 9 it can be seen that it is very close to its maximum, which for this test was ~ 0.26 . When the wind speed is increased the Power Controller reacts and adjusts the rotational speed of the rotor, and after some time the C_p -value has been returned to its maximum. It takes some time for the algorithm to recover after the increase in wind speed, however the value is maintained in the proximity of the maximum during this time. The wind gusts are reflected both in the output power and the C_p -value. The main reason to why the effect is so visible in the output power is that the wind power unit which was simulated is quite small, a bigger rotor, with higher in-

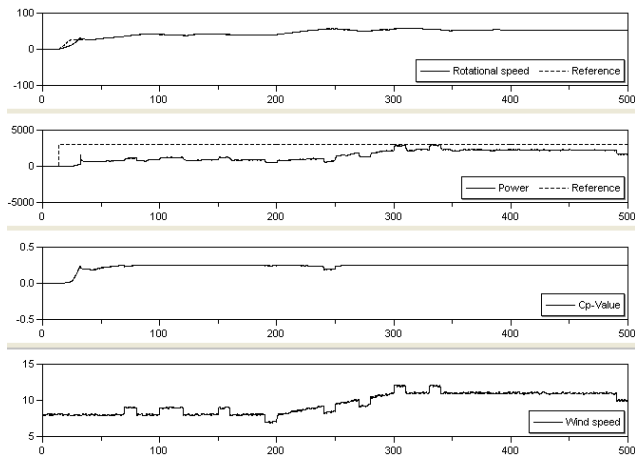


Figure 9: Simulation results, plotted from top to bottom are: **1.** The rotational speed of the turbine and its reference **2.** the power output from the unit and the calculated maximum output power **3.** the C_p -value **4.** the wind speed

ertia, would do a better job of filtering these “bumps”. The effect of the gusts on the C_p -value is quite large, however the goal of the optimizing algorithm should not be to maximize the output power over these gusts, but instead to maximize the power output during a long period of time. Otherwise the control would be forced to be very aggressive and “nervous” which is not desirable.

5.2 Active and Reactive Power Output During Operation

This case has been designed in order to test as many features of the Power Controller as possible. To do this the wind speed varies from 0 - 23 m/s during the simulation. This should cause the power controller to shut down the wind power unit both for too high and too low wind speeds. It should also optimize the power output when the wind speed is too low for nominal power and control the power output when the wind speed is sufficient. More importantly it should also confirm that the transition between the different control modes work well. The limits for low and high wind speed were chosen to 6 respectively 20 m/s.

The simulation has been divided into 7 different zones, depicted in figure 10 and 11.

1. The wind speed starts at 0 m/s and increases to 5 m/s. During this period the rotor’s rotational speed and the power reference are both zero, since the wind speed is too low for operation of the plant.

2. In this zone the wind speed is increased from 5 m/s to 10 m/s which is enough to allow operation of the unit. The power reference is raised to its nominal value, 3000 W, and the rotor starts to rotate. Since the wind speed is not high enough for nominal power the Power Controller tries to optimize the unit’s power output, as can be seen in the graph of the C_p -value in the third subplot of figure 10.
3. In this zone the wind increases further and is now high enough to allow operation at nominal effect.
4. In this zone the wind speed increases to 23 m/s, which is more than the maximum allowed for operation. Hence the power reference value is decreased to zero and the generator decelerates the rotor speed. When the rotational speed is low enough the brake is applied and the rotor stops.
5. The wind speed is now reduced to 18 m/s, which is lower than the maximum allowed wind speed for operation, and the power reference is increased to its nominal value and the rotor starts to rotate again. Since the wind speed is high enough for nominal power the Power Controller controls the power to this value.
6. The wind speed is now reduced to 11 m/s which is too low for nominal power and the Power Controller tries to optimize the unit’s output power, which can be seen on the C_p -value in the third subplot of figure 10.
7. In this zone the wind speed is decreased to 3 m/s which is well below the minimum allowed wind speed. This causes the Turbine Control to give orders to shut down the wind power unit. The power reference is set to zero and the generator decelerates the rotor. When the rotor’s rotational speed is low enough the brake is applied.

As can be seen in zone 1, 4 and 7 the Turbine Control successfully makes the decision to turn off the unit when the wind is either too high or too low for operation. The transition between the different control modes, which can be seen going from zone 2 to 3 and 5 to 6³, is working correctly. In the transition from zone 2 and 3 a small power overshoot is present before the Power Controller manages to counteract the increased wind speed. In zone 2 and 6 it can be seen that the

³in zone 2 to 3 the transition between optimization and nominal power can be observed, and in zone 5 to 6 the transition between nominal power and optimization can be observed.

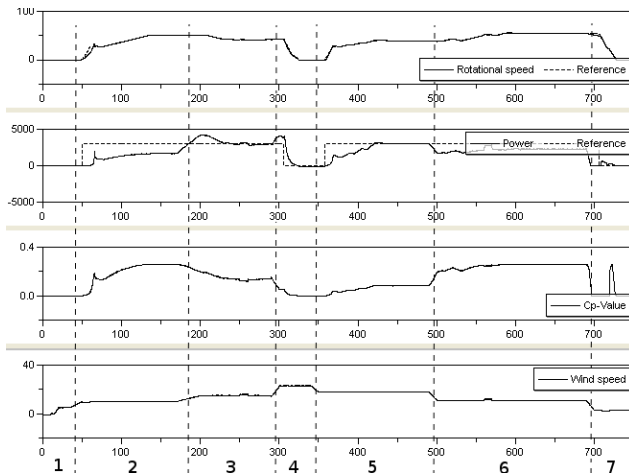


Figure 10: Simulation results, plotted for top to bottom are: **1.** the rotational speed of the turbine and its reference **2.** the power output from the unit and its reference **3.** the C_p -value **4.** the wind speed

Power Controller successfully finds the optimal operation point. In zone 3 and 5 it can be seen that the Power Controller successfully controls the output power to its nominal value, 3kW. When observing the C_p -value in zone 7 a large spike can be observed. The reason for this is that when the rotor decelerates it passes through its optimal rotational speed.

In figure 11 the results from the grid side control of the plant are presented. As expected the direct voltage is high above the nominal voltage of 230V during the time the reactive power compensation is inactive, and the reactive power output during the same time period is zero, according to Swedish grid codes [2]. The reactive power compensation is turned on after 350s, when the plant is at standstill. When the operation resumes it can be noticed that the direct voltage is controlled to its nominal value, 230V. Because of the reactive power flow the active power delivered to the point of connection is slightly lower than without reactive compensation.

5.3 Island Operation

This test case was designed to test how the plant manages to run in Island mode, that is to on its own power up and control the voltage and frequency of the AC-grid. This is done by ordering the grid connection to control the AC-voltage and frequency and the wind power unit to control the DC-voltage level.

The results from the simulation are presented in figure 12. As can be seen the power controller successfully controls the DC-level to 900V with some deviations. At the start the Plant Control orders the wind

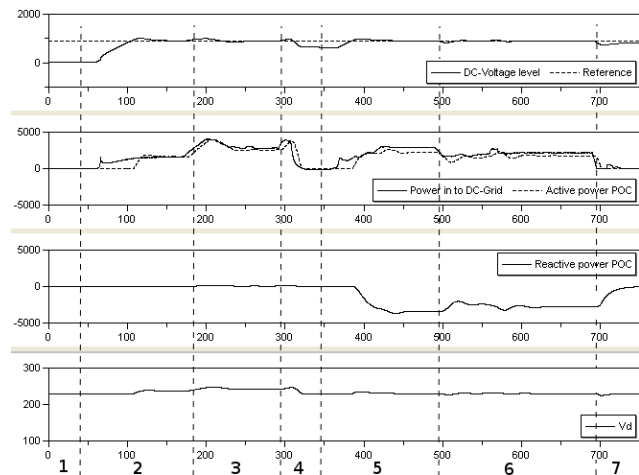


Figure 11: Simulation results, plotted from top to bottom are: **1.** the DC-voltage level and its reference **2.** the power flow into the DC-grid and active power flow out from the DC-grid **3.** the reactive power flow out from the DC-grid **4.** the direct voltage

power unit to output 1000W to power up the DC-link capacitor. When the DC-level reaches 850V the grid controller starts powering up the grid. When the power flow from the DC-grid has started the Plant Control waits until accurate power readings have been achieved and then orders the wind power unit to control the DC-voltage level. The Power Controller manages to relatively fast control the DC-voltage to 900V. After 250s the load is increased which causes the DC-voltage to start decreasing. When the DC-voltage drops the power reference is changed in order to restore the DC-voltage. As depicted in figure 12 the grid controller does not have any trouble to power up the grid, and the drop in the DC-level is not so big that it has any effect on the inverter output. The effect of the load's oscillation is visible in the DC-voltage which also is experiencing a small oscillation. However this oscillation can be neglected since it has no major effect on the system. The DC-control has no apparent problems controlling the DC-level when a varying load is connected.

6 Conclusion

This work has shown that Modelica and Dymola are powerful tools for modeling a wind power plant including power electronics. It has also been shown that it can be used to test and evaluate control algorithms before the plant is built.

All the models and control algorithms were tested using different simulations. The different simulations

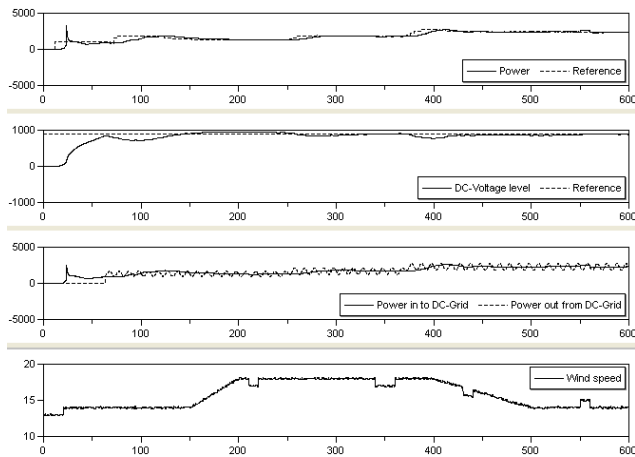


Figure 12: Simulation results, plotted from top to bottom are: **1.** the power output from the wind power unit and its reference **2.** the DC-voltage level and its reference **3.** the power in and out from the DC-grid **4.** the wind speed

tests the performance of the control algorithms during different conditions. Apart from the controls discussed some algorithms with the sole purpose of optimizing the control algorithms were also tested [6]. Unfortunately it was not possible to verify the models against real measurement data. The models are however considered to be good enough to test the control strategies. The general properties of the system are considered to be correct since the power extraction from wind power turbines is quite well documented [3][4], as well as most power electronics and generators.

During the project the performance of the control system was tested and evaluated compared to grid codes of Denmark and Sweden. Some paragraphs were not taken into account when designing the control. These grid codes would require some additional control modes, but no obvious problem in implementing that into the system was found. The details of these tests, as well as more simulation cases can be found in [6]. The performance of the control algorithms was in general good and achieved virtually all the grid codes tested.

References

[1] Energinet.dk, “Technical regulation 3.2.5 for wind power plants with a power output greater than 11 kw.” Available at <http://www.energinet.dk/SiteCollectionDocuments/Engelske20dokumenter/El/Grid20Code203.2.>

420Power20Unit20above201120kW20and20up20to201,520MW.pdf 2011-10-24.

- [2] Svenska-Kraftnät, “Technical regulation 3.2.5 for wind power plants with a power output greater than 11 kw.” Available at http://www.svk.se/Global/07_Tekniska_krav/Pdf/Foreskrifter/SvKFS2005_2.pdf 2011-10-24.
- [3] 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, Vol. 72, Iss. 1*, pp. 142–146, 2010.
- [4] A. Pintea, D. Popescu, and I. Pisica, “Robust model based control method for wind energy production,” *MCPL’2010: 5th Conference on Management and Control of Production, Coimbra : Portugal (2010)*, 2010.
- [5] W. Hu, Y. Wang, X. Song, and Z. Wang, “Development of vertical-axis wind turbine with asynchronous generator interconnected to the electric network,” *Electrical Machines and Systems, 2008. ICEMS 2008. International Conference on*, pp. 2289–2293, 2008.
- [6] J. Petersson and P. Isaksson, “Modeling and simulation of a vertical wind power plant in dymola/modelica.” Master’s thesis, Department of Industrial Electrical Engineering and Automation, Lund University, 2011, http://www.iea.lth.se/publications/MS-Theses/Full%20document/5290_full_document.pdf.
- [7] J. Thongham, 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*, pp. 1832–1837, 2009.
- [8] M. Alaküla and P. Karlsson, *Power Electronics, Devices, Converters, Control and Applications*. Department of Industrial Electrical Engineering and Automation, Lund Institute of Technology, 2010.
- [9] Modelica-Association, “Modelica®- a unified object-oriented language for physical systems modeling.” Available at <https://www.modelica.org/documents/ModelicaSpec32.pdf> 2011-08-23.

First- and Second-Order Parameter Sensitivities of a Metabolically and Isotopically Non-Stationary Biochemical Network Model*

Ralf Hannemann-Tamás^{1,5} Jana Tillack^{2,3} Moritz Schmitz¹
Michael Förster⁴ Jutta Wyes¹ Katharina Nöh^{2,3} Eric von Lieres^{2,3}
Uwe Naumann⁴ Wolfgang Wiechert^{2,3} Wolfgang Marquardt¹

¹AVT, RWTH Aachen, Germany ²IBG-1, Forschungszentrum Jülich, Germany

³JARA - High-Performance Computing ⁴STCE, RWTH Aachen, Germany

⁵MTA SZTAKI, Budapest, Hungary

Abstract

The Jülich-Aachen Dynamic optimization Environment (JADE) is employed for computing first- and second-order parameter sensitivities of a metabolically and isotopically non-stationary biochemical network model. Based on a Modelica representation of the model, code generation, algorithmic differentiation and first- and second-order adjoint sensitivity analysis are employed for computing the gradient and the Hessian of a parameter estimation objective function. In particular, we use composite adjoints, an extension of the classical adjoint sensitivity analysis, and a numerical integrator based a modification of second-order discrete adjoints of the extrapolated linearly-implicit Euler method. Therewith, the 116×116 -Hessian of the objective function with respect to 116 model parameters can be computed at the cost equivalent to only 18 objective function evaluations, while computing the same Hessian with the cheapest finite-difference formula would require 6845 evaluations of the objective function.

Keywords: biochemical network model; parameter sensitivities; automatic differentiation

1 Introduction

Kinetic-based modeling is the method of choice for unraveling complex interactions in living microorganisms [8]. Only this approach allows to analyze the response of organisms to extracellular stimuli, such

as changes in the substrate concentration. Moreover, industrial processes are typically run in cultivation modes, in which the intracellular metabolism cannot be assumed to be in a stationary state. Metabolically non-stationary network models include rate laws for the enzyme catalyzed reactions, and the corresponding model equations depend on several kinetic parameters. The rate laws also include regulatory effects, i.e. activation and inhibition by other metabolites, which increases the overall complexity of the network. Kinetic models are normally calibrated using measured intracellular metabolite concentrations. However, the ratio between the number of unknown parameters and the quantity of available measurement data is often insufficient. Consequently, the kinetic parameters are poorly determined or even non-identifiable on the basis of such data.

This limitation can be overcome by combining classical kinetic modeling with an isotope-labeling approach ([11], [3]). In this approach, experiments are performed with a specifically ¹³C-labeled substrate instead of the slightly lighter, naturally ¹²C-labeled substrate. ¹²C as well as ¹³C-atoms are distributed through the reaction network and form specific labeling signatures in the involved metabolites. These signatures, so called isotopologues, consist of differently many heavier (labeled) and lighter (naturally labeled) carbon atoms, and can be quantified using the LC-MS measurement technique [12]. Hence, the use of labeled substrates increases the amount of data for each metabolite proportional to its number of carbon atoms. However, the model dimensions are strongly increased. The extended model requires increased computational resources not only for solving the forward problem, but also for determining gradient and

*This work was carried out during the tenure of an ERCIM “Alain Bensoussan” fellowship program. This program is supported by the Marie Curie co-funding of regional, national and international programs (COFUND) of the European Commission.

Hessian information for efficiently solving the inverse parameter estimation problem.

2 Biochemical Network Model

The combined metabolically and isotopically non-stationary modeling approach is illustrated with an example network of *Escherichia coli* [2]. The biochemical network covers glycolysis and the pentose phosphate pathway. The model involves 28 metabolites (thereof 8 co-metabolites) and 28 reactions (thereof 8 effluxes), as illustrated in Figure 1.

The equations for the kinetic rates, r , the values of the stoichiometric constants, p_{stoich} , and of the kinetic parameters, p_{kin} , and the initial metabolite concentrations, c_0 , are taken from the same publication [2]. The model is extended from the metabolically non-stationary case to the metabolically and isotopically non-stationary case by transforming the differential equations, that describe the change of metabolite concentrations, c , over time (Equation 1), into sets of differential equations for the so-called cumomers, m (Equation 2).

$$\begin{aligned} \frac{dc}{dt} &= f(c, r, p_{stoich}) \\ r &= g(c, p_{kin}) \\ c(0) &= c_0 \end{aligned} \quad (1)$$

A cumomer can be interpreted as a molecule fragment that is fully labeled to a specified degree ([13], [14]). The cumomer, e.g., $m_{i,j}$ of a metabolite, m , with three carbon atoms includes the four differently labeled species $m_{i,j} = \sum_{i,j \in \{0,1\}} m_{i,j}$, namely m_{00} , m_{10} , m_{01} and m_{11} , where the digits 0 and 1 denote the isotopes ^{12}C and ^{13}C , respectively. The concentration of a cumomer fraction is defined as the sum of the concentrations of all corresponding species. In particular, the concentration of the cumomer m_{xxx} is the absolute metabolite concentration, c . A metabolite with n carbon atoms has 2^n cumomers in total. The formulation of cumomer balance equations requires structural information on: (1) the underlying metabolic network model, i.e. all participating enzymatic reaction steps, (2) the carbon atom transitions for each of these steps (see Figure 2 for an example), and (3) the kinetic mechanisms [11].

$$\begin{aligned} \frac{dm}{dt} &= f(m, r, p_{stoich}) \\ m(0) &= m_0 \end{aligned} \quad (2)$$

The cumomer balances in Equation 2 are combined with the original kinetic equations from Equation 1.

The vector c , containing all metabolite concentrations, is a subset of the vector m , containing all cumomer fractions m_{ijk} with $i, j, k \in \{1, x\}$ of all metabolites. The initial values of the algebraic variables, r , are determined such as to fulfill the algebraic equation, g .

Realistic models, e.g., of the central carbon metabolism, have around 30-40 metabolites, 50-60 reactions and 30-40 regulatory relations leading to model dimensions of 1,000 to 10,000 equations. Moreover, Equation 2 is typically stiff, dense and highly non-linear.

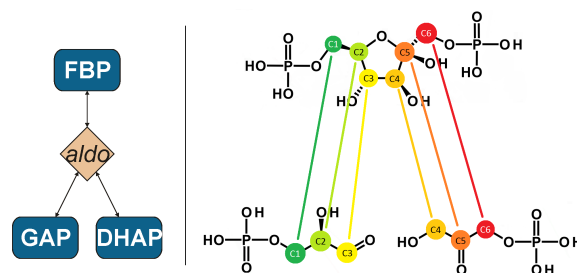


Figure 2: Carbon atom transition of a reaction that converts D-fructose-1,6-bisphosphate (FBP) into glyceraldehyde-3-phosphate (GAP) and dihydroxyacetone-phosphate (DHAP). The lines describe transitions of individual carbon atoms from the substrate to the product.

The final *E. coli* network model contains 682 differential equations that are linear combinations of the non-linear rate equations (see Equation 2). The rates do generally not only depend on the concentrations of the related substrate and product molecules, but can also depend on the concentrations of other molecules that act as activators and inhibitors of the catalyzed reaction. Equations 3 and 4 show typical examples in which the kinetic parameters are highlighted in bold-face. Sensitivities of the model solution with respect to these parameters are often required for parameter estimation and in the context of metabolic control analysis.

Equation 3 describes the enzyme D-glucose-6-phosphate aldose-ketose-isomerase (*pgi*) and is formally a reversible Michaelis-Menten kinetic with one generic inhibitor. Parameters are the maximal reaction rate r^{max} , an equilibrium constant k_{eq} , two inhibition constants k_i , and two affinity constants k_m . Equation 4 describes the enzyme phosphoglycerate kinase (*pgk*) and is formally a two-substrate reversible Michaelis-Menten kinetic. Parameters are, in addition to the first kinetic equation, the coupling constants of the co-metabolites ATP and ADP.

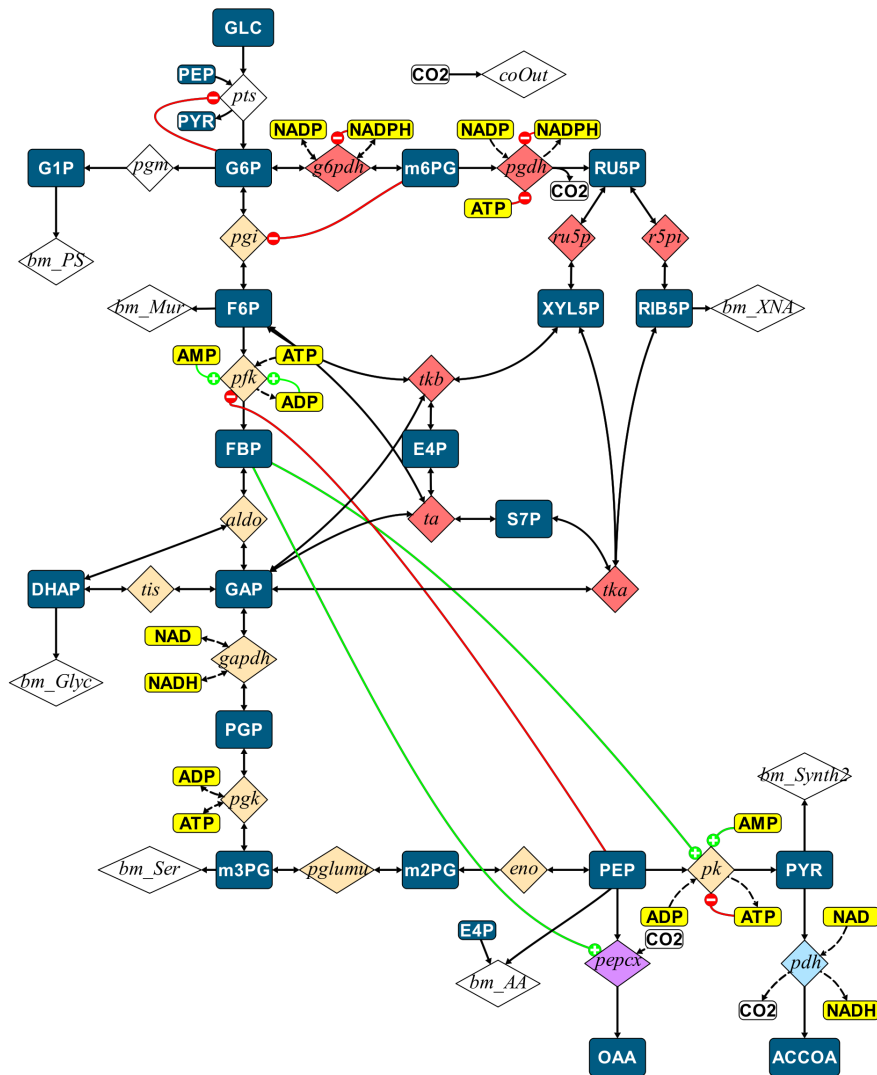


Figure 1: Biochemical network of *E. coli* including the glycolysis (orange) and the pentose phosphate pathway (red). The metabolites (rectangles) are converted by reactions (rhombi). Additional lines show regulatory interactions: activation (green lines), inhibition (red lines) and co-metabolite coupling (dashed lines, yellow metabolites).

3 JADE

The biochemical network model from the previous section has been implemented in Modelica and tested with Dymola. However, Dymola does not provide capabilities for higher-order sensitivity analysis, which are essential for many engineering tasks such as parameter estimation, optimal experimental design, optimal control and dynamic real-time optimization (DRTO). This gap will be closed by the *Jülich Aachen Dynamic Optimization Environment* (JADE), a new research program that sustains ongoing collaborations between Aachener Verfahrenstechnik – Process Systems Engineering (AVT.PT), the Jülich Biotechnology Institute (IBG-1), and Software Tools for Computa-

tional Engineering (STCE). AVT.PT and STCE are both chairs at RWTH Aachen University and IBG-1 belongs to the Forschungszentrum Jülich. The JADE concept includes a software infrastructure for sensitivity analysis of differential-algebraic equation systems.

This publication addresses a prototypical task within the JADE framework, the determination of parameter sensitivities of a residual function for estimating unknown model parameters. The biochemical network example is taken as an example, but the presented software infrastructure works for a wider class of Modelica models, without discontinuous elements, i.e. without “if”- and “when”-assignments. A software infrastructure is presented, that provides an easy-to-use integrated solution for determining the required

$$r_{pgi} = \frac{r_{pgi}^{\max} \left(c_{G6P} - \frac{c_{F6P}}{k_{eq,pgi}} \right)}{km_{G6P,pgi} \left(1 + \frac{c_{F6P}}{km_{F6P,pgi} \left(1 + \frac{c_{m6PG}}{ki_{F6P,m6PG,pgi}} \right)} \right) + c_{G6P}} \quad (3)$$

$$r_{pgk} = \frac{r_{pgk}^{\max} \left(c_{ADP} \cdot c_{PGP} - \frac{c_{ATP} \cdot c_{m3PG}}{k_{eq,pgk}} \right)}{\left(km_{ADP,pgk} \left(1 + \frac{c_{ATP}}{km_{ATP,pgk}} \right) + c_{ADP} \right) \left(km_{PGP,pgk} \left(1 + \frac{c_{m3PG}}{km_{m3PG,pgk}} \right) + c_{PGP} \right)} \quad (4)$$

first- and second-order derivatives.

Workflow

The workflow for computing sensitivity information can be divided in three layers (see Figure 3 for a schematic sketch):

1. A so-called equation set object (ESO), an instance of a C++ class which provides data and methods related to the model.
2. A Meta ESO object, an instance of a C++ class which assembles one ESO or, in the case of multistage models, several ESOs and information about the parametrization of the model (we refer to [9, 10] for details on multistage problems).
3. Drivers for the NIXE integrator [5], a numerical solver for (adjoint) sensitivity analysis of DAEs, based on the information assembled in the Meta ESO, to carry out sensitivity analysis tasks.

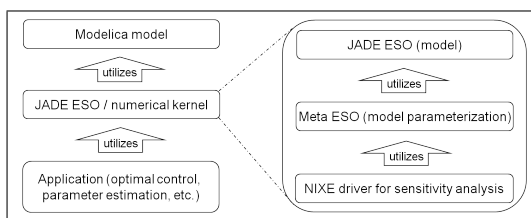


Figure 3: Layers of the software infrastructure.

Currently, *flat* Modelica models are translated into a subset of the C language, referred to as C-, by means of the Mof2C- application. In *flat* or *flattend* Modelica models, all object-oriented features are removed by the expansion of all sub-models and their connections. In particular, a flat Modelica model contains no sub-model, it has a “*flat* hierarchy”. A residual function of the DAE is created to be differentiated by means of algorithmic differentiation in form of the derivative code compiler (dcc) [7], an AD tool relying on semantic source code transformation. On Windows platforms,

the source code, generated by Mof2C- and dcc is compiled into a dynamic link library. Figure 4 shows a typical workflow within the JADE framework.

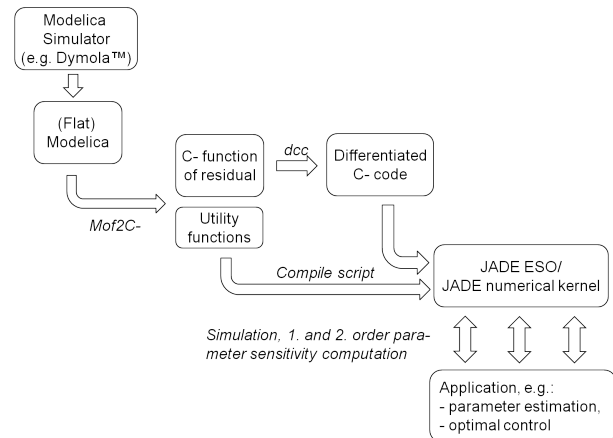


Figure 4: JADE workflow for sensitivity analysis.

4 Results

We present first- and second-order adjoint sensitivity computations for the biochemical network model from section 2. The model is formulated in Modelica without using discontinuous elements. It belongs to the class of smooth semi-explicit index-1 differential algebraic equations of the type of Equations 5 to 7.

$$\dot{x}(t, p) = f(x(t, p), y(t, p), p), \quad t \in [t_0, t_f], \quad (5)$$

$$0 = g(x(t, p), y(t, p), p), \quad t \in [t_0, t_f], \quad (6)$$

$$x(t_0, p) = x_0, \quad (7)$$

Here, $x(t, p) \in \mathbb{R}^{n_x}$ and $y(t, p) \in \mathbb{R}^{n_y}$ denote the vectors of differential and algebraic state variables, $p \in \mathbb{R}^{n_p}$ is the parameter vector, f and g denote the differential and algebraic equations, respectively, $x_0 \in \mathbb{R}^{n_x}$ is the vector of initial values and t_0 and t_f are the initial and final times, respectively.

The model comprises 1488 state variables, thereof 683 differential and 805 algebraic, as well as 337 parameters, thereof 116 relevant for a typical parameter

estimation. The model is sparse in that the Jacobians of f and g with respect to x , y and p have in the sum only 9121 nonzero entries. The initial time is set to $t_0 = -20$ in order to simulate the system in a stationary state before a concentration pulse is applied at $t = 0$, and the final time is $t_f = 40$.

For the purpose of parameter estimation we need to compute a least-squares residual, as well as its gradient and Hessian. Let $y_{out}(t, p) = (y_{i_1}(t, p), \dots, y_{i_{n_{y,out}}}(t, p)) \in \mathbb{R}^{n_{y,out}}$, $i_j \in \{1, \dots, n_y\}$, $j = 1, \dots, n_{y,out}$, denote the vector of measured variables, which is in the present example a subset of the algebraic variables. For the residual we consider a finite time series $t_1 < t_2 < \dots < t_N$ and a matrix of corresponding measurements $\tilde{Y} = (\tilde{y}_{ij}) \in \mathbb{R}^{N \times n_{y,out}}$. With scalar weights σ_{ij} , $i = 1, \dots, N$, $j = 1, \dots, n_{y,out}$, the parameter estimation objective function has the following form:

$$\begin{aligned} \Phi(p) &= \phi(y_{out}(t_1, p), y_{out}(t_2, p), \dots, y_{out}(t_N, p)) \\ &:= \sum_{i=1}^N \sum_{j=1}^{n_{y,out}} \sigma_{ij} (y_{out,j}(t_i, p) - \tilde{y}_{ij})^2. \end{aligned} \quad (8)$$

We assume measurements to be available for $n_{y,out} = 103$ output variables every 0.5 seconds, starting from $t_1 = 0$ to $t_N = t_{81} = 40$. As real measurements are currently not available, synthetic data $\tilde{Y} = (\tilde{y}_{ij}) \in \mathbb{R}^{81 \times 103}$ were generated by adding normally distributed noise with a standard deviation of 10% to the nominal values. The weights are chosen as:

$$\sigma_{ij} = \frac{1}{0.01 + \tilde{y}_{ij}^2}, \quad i = 1, \dots, 81, \quad j = 1, \dots, 103,$$

The summand 0.01 in the denominator is introduced for avoiding division by zero in the case $\tilde{y}_{ij} = 0$ and to reduce the impact of small-valued measurements.

Let $p_{est} \in \mathbb{R}^{n_{p,est}}$ denote the vector of parameters to be estimated: $p_{est,j} = p_{i_j}$, $i_j \in \{1, \dots, n_p\}$, $j = 1, \dots, n_{p,est}$, $n_{p,est} = 116$. Our software infrastructure is benchmarked for the following tasks:

1. Simulate the original model
2. Compute value of the objective function Φ .
3. Compute the gradient $\partial\Phi/\partial p_{est}$ by means of first-order adjoint sensitivity analysis.
4. Compute the gradient $\partial\Phi/\partial p_{est}$ by means of first-order forward sensitivity analysis.
5. Compute the Hessian $\partial^2\Phi/\partial p_{est}^2$ by means of second-order adjoint sensitivity analysis.

Code Generation and Compilation

All computations are performed on a Notebook with a 2.53 MHz Intel Core2 SP9600 processor, equipped with 4 GB RAM and running Linux Mint 12.

As illustrated in Figure 4, the first task of the JADE architecture is to generate C-code from a flat Modelica model. This done by the Mof2C- compiler, which generates a C-function of the model residual and related utility functions, e.g., for providing access to the variable names. This part of the code generation takes roughly 4 seconds. Then, the derivative code compiler dcc, an algorithmic differentiation (AD) tool, is applied for generating derivatives of the model residual. This part takes approximately 5 minutes, thereof 4 minutes for the generation of the second-order adjoints of the model residuals.

The generated code, including the derivative codes, is then compiled either in a dynamic link library (DLL) on Windows platforms or a shared object on Linux or UNIX platforms. Here, the compilation times strongly depend on the compiler flags, especially on the optimization flags. The sequential compilation times with the g++-4.6.1 compiler of the GNU Compiler Collection (gcc) are 2 minutes (thereof 1 minute for the second-order adjoints) for non-optimized code, and for optimized code (-O3-flag) 53 minutes (thereof 37 minutes for the second-order adjoints).

Simulation and Sensitivity Analysis

We apply the JADE infrastructure for simulating and evaluating the objective function, as well as its gradient and Hessian with either optimized or non-optimized compiled code. The numerical kernel relies on the NIXE integrator. NIXE implements the extrapolated linearly-implicit Euler method, and provides facilities for higher-order forward or adjoint sensitivity analysis. In detail, NIXE implements a modified discrete adjoint method for the adjoint sensitivity analysis [5]. Further, since the objective function ϕ in Equation 8 depends on different points in time, we use the technique of *composite adjoints* [4], instead of the classical adjoint sensitivity analysis (which only submits one final time) [1]. Whenever the gradient or Hessian of a DAE-embedded functional of the type $\phi(x(t_1, p), \dots, x(t_N, p))$ with respect to sufficient many parameters has to be computed (cf. Equation 8), from the view of computational efficiency, composite adjoints are the method of choice. Roughly spoken, composite adjoints compute a linear combination of the N classical adjoints associated

with $\phi(x(t_1, p), \dots, x(t_N, p))$ corresponding to the final times t_1, \dots, t_N . The computational cost of composite adjoints is equivalent to the cost of only one classical adjoint computation with a final condition at t_N . For details we refer to [4].

Table 1 shows the performance of different computations. For comparison, we have also executed a simulation with Dymola 7.1 in combination with MS Visual Studio 2008 on the same notebook but running Windows 7 (see last row of Table 1).

Table 1: Computational performance

JADE results, AbsTol=RelTol= 10^{-5} 1488 state variables, 116 parameters		
Task	Run time	
	Optimized	Non-opt.
Simulation	1.7 s	2.3 s
Objective	10.5 s	14.5 s
Gradient (adjoint)	14.5 s	19.9 s
Gradient (forward)	46.8 s	63.5 s
Hessian (2nd adjoints)	180.0 s	465.0 s
Dymola Simulation	1.6 s (DASSL, Tol= 10^{-5})	

The simulation time of JADE is competitive with Dymola, for both the optimized and the non-optimized compiled codes. However, Dymola does neither support first-order nor second-order sensitivity analysis. We observe that the evaluation the objective function takes much longer than the simulation. This is due to the NIXE integrator stopping at the measurement times and resetting the adaptive step size control. Computing the 116 components of the gradient with adjoint sensitivity analysis takes only about 1.5 times the time of one single function evaluation for both the optimized and the non-optimized codes. Forward sensitivity analysis is 3 times slower (optimized and non-optimized). Computing the 116×116 -Hessian matrix takes 180 seconds with the optimized compiled code and 465 seconds with the non-optimized compiled code.

Comparison with Finite Differences

If we compare the computational times of the JADE sensitivity analysis with the costs of finite differences, we clearly see the superiority of the tailored numerical methods of JADE. Table 2 shows compute time ratios of the different sensitivity tasks as compared to a single objective function evaluation.

The cheapest finite differences formulas would require $117 = 1 + 116$ function evaluations for the gradient and $6845 = 1 + 116 + 116^2/2$ function evaluations

for the Hessian. The excellent numerical performance of the JADE prototype is mainly achieved by combining the AD tool dcc [7] with composite adjoints [4] that are computed with the specifically tailored numerical integrator NIXE [5], which strongly exploits the structure of the underlying (adjoint) sensitivity equations.

Conclusions

We have introduced the JADE platform for first- and second-order sensitivity analysis of DAE models. The platform combines code generation, algorithmic differentiation and a customized numerical integrator for forward and adjoint sensitivity analysis. The presented results in particular for computing the Hessian of the studied parameter estimation objective function are more than competitive. The complete 116×116 Hessian of the objective function is computed at the cost of 18 single function evaluations, yielding accurate second-order derivatives. In comparison, computing the same Hessian with the cheapest and least accurate finite difference formula would require 6845 function evaluations. This makes the JADE platform particularly attractive for large-scale applications with nonlinear numerical optimization solvers that require second-order derivatives.

Outlook

Up to now, the numerical methods of JADE are restricted to smooth Modelica models without discontinuities. However, many systems, e.g., from engineering or biotechnology, need to be modeled with non-smooth differential-algebraic equations. In addition, the modeling process can yield under-determined differential-algebraic systems (more variables than equations). In this case, some of the model variables must be determined by external criteria, for example by means of an optimization criterion. The resulting models do not belong to the well-known class of hy-

Table 2: JADE (optimized) versus finite differences

Task	Cost factor = $\frac{\text{run_time}(\text{Task})}{\text{run_time}(\text{Objective})}$		
	JADE forward	JADE adjoint	Finite differences
Objective	1	-	1
Gradient	4.5	1.4	117
Hessian	-	17.2	6845

brid DAE systems, but a novel class of *non-smooth DAEO* systems can be defined, where the “O” denotes optimization. The concept of the JADE prototype, i.e. combining a high-level model language like Modelica with algorithmic differentiation and tailored numerical solution methods, will be extended to the classes of non-smooth DAE and DAEO systems.

References

- [1] Cao, Y, Li S, Petzold L, Serban R. Adjoint sensitivity analysis for differential-algebraic equations: The adjoint DAE system and its numerical solution. *SIAM Journal On Scientific Computing* 24(3):1076–1089, 2003.
- [2] Chassagnole C, Noisommit-Rizzi N, Schmid J W, Mauch K, Reuss M. Dynamic modeling of the central carbon metabolism of escherichia coli. *Biotechnol Bioeng*, 79(1):53–73, 2002.
- [3] de Graaf A A, Maathuis A, de Waard P, Deutz N E P, Dijkema C, de Vos W M, Venema K. Profiling human gut bacterial metabolism and its kinetics using [u-13c]glucose and nmr. *NMR Biomed*, 23(1):2–12, 2010.
- [4] Hannemann R, and Marquardt W. Continuous and discrete composite adjoints for the Hessian of the Lagrangian in shooting algorithms for dynamic optimization. *SIAM Journal On Scientific Computing*, 31(6): 4675–4695, 2010.
- [5] Hannemann R, Marquardt W, Gendler B, Naumann U. Discrete first- and second-order adjoints and automatic differentiation for the sensitivity analysis of dynamic models. *Procedia Computer Science*, 1(1):297–305, 2010.
- [6] Modelica Association. *Modelica® - A Unified Object-Oriented Language for Physical Systems Modeling. Language Specification. Version 3.2*. Linköping, Sweden: Modelica Association, 2010.
- [7] Naumann, U. *The Art of Differentiating Computer Programs - An Introduction to Algorithmic Differentiation*, Volume 24 of *Software, environments, tools*. SIAM, 2012.
- [8] Steuer R. Exploring the dynamics of large-scale biochemical networks: A computational perspective. *The Open Bioinformatics Journal*, 5:4–15, 2011.
- [9] Vassiliadis V, Sargent R, Pantelides C. Solution of a class of multistage dynamic optimization problems. 1. Problems without path constraints. *Ind Eng Chem Res*, 33(9):2111–2122, 1994.
- [10] Vassiliadis V, Sargent R, Pantelides C. Solution of a class of multistage dynamic optimization problems. 2. Problems with path constraints. *Ind Eng Chem Res*, 33(9):2123–2133, 1994.
- [11] Wahl S A, Noeh K, Wiechert W. ¹³C labeling experiments at metabolic nonstationary conditions: an exploratory study. *BMC Bioinformatics*, 9:152, 2008.
- [12] Wiechert W. ¹³C metabolic flux analysis. *Metab Eng*, 3(3):195–206, 2001.
- [13] Wiechert W, Moellney M, Isermann N, Wurzel M, de Graaf A A. Bidirectional reaction steps in metabolic networks: III. explicit solution and analysis of isotopomer labeling systems. *Biotechnol Bioeng*, 66(2):69–85, 1999.
- [14] Wiechert W, Wurzel M. Metabolic isotopomer labeling systems. Part I: global dynamic behavior. *Math Biosci*, 169(2):173–205, 2001.

Collocation Methods for Optimization in a Modelica Environment

Fredrik Magnusson^a Johan Åkesson^{a,b}

^aDepartment of Automatic Control, Lund University, Sweden

^bModelon AB, Lund, Sweden

Abstract

The solution of generic dynamic optimization problems described by Modelica, and its extension Optimica, code using direct collocation methods is discussed. We start by providing a description of dynamic optimization problems in general and how to solve them by means of direct collocation. Next, an existing implementation of a collocation algorithm in JModelica.org, using CasADi and IPOPT, is presented. The extensions made to this implementation are reported.

The new implementation is compared to an old C-based collocation algorithm in JModelica.org in two benchmarks. The presented benchmarks are based on a continuously stirred tank reactor and a combined cycle power plant. The new algorithm and its surrounding framework is more flexible and shown to be several times more efficient than its predecessor.

Keywords: dynamic optimization; JModelica.org; collocation; nonlinear programming; CasADi

1 Introduction

Optimization of large-scale dynamic systems is becoming a standard industrial technology. Applications include minimization of material and energy consumption during set-point transitions in power plants and chemical processes, minimizing lap times for vehicle systems and trajectory optimization in robotics.

There are different kinds of dynamic optimization problems and in this paper we consider two categories. The first is optimal control, where the aim is to find control variable trajectories (and possibly parameters) that minimize, for example, the amount of resources spent to perform a specified action. The second category is parameter estimation, where the problem is to find the values of unknown model parameters that

allow the model to behave according to some given measurement data.

Solving dynamic optimization problems is useful in many different fields and applications. Parameter estimation is used to improve physical models in general. Optimal control has many applications, in both on-line and off-line settings. On-line optimal control is usually done in the form of model predictive control. Off-line applications include finding optimal trajectories for the transition between two stationary operating conditions in a system, which can be used either as a reference during manual control or as a target for automatic control if combined with feedback. Another example is the identification of system bottlenecks, for example by analyzing adjoint variables.

There are many approaches to solving dynamic optimization problems. Until the 1970s, problems were typically solved using dynamic programming or Pontryagin's maximum principle. These approaches are ill-suited for large-scale problems and have trouble handling inequality constraints. Modern techniques often involve finding an approximate solution to the infinite-dimensional optimization problem by transcribing it into a finite-dimensional nonlinear program (NLP). These are called direct methods. The main difference among direct approaches is how to handle the constraints describing the system dynamics. In this paper, direct collocation is used. Another common approach is direct multiple shooting. See [1] and [2] for overviews on different direct methods.

JModelica.org [3] is a tool targeting large-scale dynamic optimization. The system dynamics are described using Modelica, and the optimization formulation is done with the use of the Modelica extension Optimica [4]. In this paper, we implement an optimization algorithm in JModelica.org for solution of dynamic optimization problems described by Modelica and Optimica code. This work is a continuation of the work begun in [5], where CasADi and JModelica.org were integrated and a prototypical collocation method was implemented based on this integra-

This work was supported by the Swedish Research Council through the LCCC Linnaeus Center. We would also like to thank Francesco Casella for letting us use the combined cycle power plant model.

tion. This prototype has since been refined and extended to support additional problem formulations and solution techniques. Additional benchmarks have also been performed, as reported in [6].

The outline of the paper is as follows. In Section 2, a general class of dynamic optimization problems is presented. In Section 3, we discuss how to solve this class of problems using direct collocation. In Section 4, the prominent tools used to implement the described collocation method in a Modelica environment are presented. In Section 5, we present the extensions made to the implementation from previous work. In Section 6, the implemented algorithm is compared to a similar existing algorithm. The two algorithms are applied to a continuously stirred tank reactor and to a combined cycle power plant. Finally, in Section 7, the paper is summarized and some future work is discussed. The work presented in this paper is a result of [6], where additional details are available.

Throughout the paper, the following notation is used. The integer interval from $a \in \mathbb{Z}$ to $b \in \mathbb{Z}$ is denoted by $[a..b]$. All kinds of products between scalars, vectors and matrices are denoted by the binary operator \cdot . The space of functions continuous of order k from \mathbb{R}^m into \mathbb{R}^n is denoted by $C^k(\mathbb{R}^m, \mathbb{R}^n)$, where $k = -1$ means that the functions may be discontinuous. No distinction between tuples and vectors is made.

2 Dynamic optimization

We consider systems whose dynamics are described by a single and fully implicit differential algebraic equation (DAE) system of index one (or zero). That is, an equation system of the form

$$F(t, \dot{x}(t), x(t), u(t), w(t), p) = 0,$$

where $t \in \mathbb{R}$ is the sole independent variable: time, $x \in C^0(\mathbb{R}, \mathbb{R}^{n_x})$ is the state, $u \in C^{-1}(\mathbb{R}, \mathbb{R}^{n_u})$ is the vector-valued control variable, $w \in C^{-1}(\mathbb{R}, \mathbb{R}^{n_w})$ is the vector-valued algebraic variable and $p \in \mathbb{R}^{n_p}$ is the vector of parameters to be optimized, that is, the free parameters. Initial conditions are also given on a fully implicit form, i.e.,

$$F_0(\dot{x}(t_0), x(t_0), u(t_0), w(t_0), p) = 0,$$

where t_0 is the start time. For ease of notation, we compose the time-dependent variables into a single variable z , that is,

$$z := (\dot{x}, x, u, w).$$

The system dynamics are thus fully described by

$$\begin{aligned} F(t, z(t), p) &= 0, \quad \forall t \in [t_0, t_f], \\ F_0(z(t_0), p) &= 0, \end{aligned}$$

where t_f is the final time and

$$\begin{aligned} F &\in C^2(\mathbb{R} \times \mathbb{R}^{n_z} \times \mathbb{R}^{n_p}, \mathbb{R}^{n_x+n_w}), \\ F_0 &\in C^2(\mathbb{R}^{n_z} \times \mathbb{R}^{n_p}, \mathbb{R}^{n_x}), \\ n_z &:= 2 \cdot n_x + n_u + n_w. \end{aligned}$$

These continuity requirements, and some of the continuity requirements stated later in this section, are needed to establish the second-order optimality conditions and also to find a solution to the first-order optimality condition using some variation of Newton's method.

The general problem studied in this paper is to

$$\text{minimize} \quad f(t_0, t_f, z, p), \quad (1a)$$

$$\text{with respect to} \quad t_0, t_f, z, p,$$

$$\text{subject to} \quad F(t, z(t), p) = 0, \quad (1b)$$

$$F_0(z(t_0), p) = 0, \quad (1c)$$

$$z_L \leq z(t) \leq z_U, \quad (1d)$$

$$p_L \leq p \leq p_U, \quad (1e)$$

$$g_e(t_0, t_f, t, z(t), p) = 0, \quad (1f)$$

$$g_i(t_0, t_f, t, z(t), p) \leq 0, \quad (1g)$$

$$G_e(t_0, t_f, Z_e, p) = 0, \quad (1h)$$

$$G_i(t_0, t_f, Z_i, p) \leq 0, \quad (1i)$$

$$\forall t \in [t_0, t_f].$$

The objective (1a) can take on many forms. For optimal control problems, it is typically a Bolza functional, that is, a function on the form

$$\begin{aligned} f(t_0, t_f, z, p) &= \phi(t_0, z(t_0), t_f, z(t_f), p) + \\ &\int_{t_0}^{t_f} L(t, z(t), p) dt, \end{aligned} \quad (2)$$

where

$$\phi \in C^2(\mathbb{R} \times \mathbb{R}^{n_z} \times \mathbb{R} \times \mathbb{R}^{n_z} \times \mathbb{R}^{n_p}, \mathbb{R})$$

is called the Mayer term and

$$L \in C^2(\mathbb{R} \times \mathbb{R}^{n_z} \times \mathbb{R}^{n_p}, \mathbb{R})$$

is called the Lagrange integrand.

For parameter estimation, the objective function is typically formulated using a weighted least squares sum, penalizing the deviation of the measured variables from the discrete measurement data. However,

in this paper we choose a slightly different approach. We first interpolate the discrete measurement data to form $y_m \in C^0(\mathbb{R}, \mathbb{R}^{n_y})$, where n_y is the number of measured variables. This function gives the approximated trajectories for the vector-valued measured variable $y \in C^{-1}(\mathbb{R}, \mathbb{R}^{n_y})$. Any of the states, algebraic variables and control variables can be measured variables. The objective is then chosen as a continuous weighted least squares function, given by

$$f(z, p) = \int_{t_0}^{t_f} (y(t) - y_m(t)) \cdot Q \cdot (y(t) - y_m(t)) dt, \quad (3)$$

where $Q \in \mathbb{R}^{n_y \times n_y}$ is the weighting matrix. The reason for this approach is discussed in Section 3.

The constraints (1b) and (1c) enforce the system dynamics and initial conditions. The constraints (1d) and (1e) are variable bounds, which are enforced during the entire time horizon $[t_0, t_f]$, where $z_L \in (\mathbb{R} \cup \{-\infty\})^{n_z}$ and $p_L \in (\mathbb{R} \cup \{-\infty\})^{n_p}$ are the lower bounds and $z_U \in (\mathbb{R} \cup \{\infty\})^{n_z}$ and $p_U \in (\mathbb{R} \cup \{\infty\})^{n_p}$ are the upper bounds. The constraints (1f) and (1g) are called path constraints. These can for example be used to describe that a vehicle must follow a certain path. Finally, the constraints (1h) and (1i) are called point constraints. These are similar to the path constraints, with the difference being that they are only enforced at specific time points, rather than during the entire time horizon. The vectors Z_e and Z_i contain the variable values at all the time points used in the point constraints, i.e.

$$Z_e = (z(T_1), z(T_2), \dots, z(T_m)),$$

where T_i is the time point at which point constraint i is enforced and m is the number of constraint points. A typical example of a point constraint is terminal constraints, where variable values are specified at the end of the time horizon. The path constraint functions g_e and g_i as well as the point constraint functions G_e and G_i must be twice continuously differentiable.

The general problem formulation (1) covers a large class of problems. The constraints (1d) to (1i) are optional, whereas the constructs in (1a) to (1c) are required to get a well-posed problem. The start and final time can be either fixed or free. For example, letting the final time be free and choosing the cost function as $f(t_0, t_f, z, p) = t_f$ allows for the formulation of minimum time problems, where the goal is to minimize the time required to perform some action, often specified in the form of terminal constraints.

A possible generalization of (1) is the division of the time horizon into multiple phases, where at the

phase boundaries the DAE system is allowed to change and/or the states may be discontinuous. Another possible generalization is enforcing continuity for control and algebraic variables and then including their respective derivatives in the constraints and cost function. These generalizations are however outside the scope of this paper.

3 Collocation methods

3.1 Collocation polynomials

We will now describe how to solve the dynamic optimization problem (1) by means of direct collocation, using an approach similar to the ones described in [1] and [7]. The time horizon is discretized into n_e elements, and within element i the time-dependent variable z is approximated using a vector-valued polynomial $z_i = (\hat{x}_i, x_i, u_i, w_i)$, called a collocation polynomial. In element i , the time is normalized according to

$$\tilde{\tau}_i(\tau) = t_{i-1} + h_i \cdot (t_f - t_0) \cdot \tau, \quad \tau \in [0, 1], \quad \forall i \in [1..n_e], \quad (4)$$

where t_i is the time at the end of element i , which is called the mesh point of element i , and h_i is the length of element i . The element lengths have been normalized so that the sum of all lengths equals 1. This normalization facilitates the optimization of t_f and t_0 by keeping element lengths constant.

The collocation polynomials are formed by choosing a number n_c of collocation points (which is assumed to be the same for each element). Let $\tau_{i,k}$ denote collocation point k in element i , and let $z_{i,k} = (\hat{x}_{i,k}, x_{i,k}, u_{i,k}, w_{i,k})$ denote the value of $z(\tau_{i,k})$. The collocation polynomials are then formed using Lagrange interpolation polynomials, using the collocation points as interpolation points. Since the states need to be continuous even at the element boundaries, we introduce an additional interpolation point at the start of each element for the state collocation polynomials, denoted by $\tau_{i,0} := 0$. We thus get the collocation polynomials

$$\begin{aligned} x_i(\tau) &= \sum_{k=0}^{n_c} x_{i,k} \cdot \tilde{\ell}_k(\tau), \\ u_i(\tau) &= \sum_{k=1}^{n_c} u_{i,k} \cdot \ell_k(\tau), \\ w_i(\tau) &= \sum_{k=1}^{n_c} w_{i,k} \cdot \ell_k(\tau), \end{aligned} \quad (5)$$

where $\tilde{\ell}_k$ and ℓ_k are the Lagrange basis polynomials, respectively with and without the additional interpola-

tion point $\tau_{i,0}$. The basis polynomials are given by

$$\begin{aligned}\tilde{\ell}_k(\tau) &= \prod_{l \in [0..n_c] \setminus \{k\}} \frac{\tau - \tau_l}{\tau_k - \tau_l}, \quad \forall k \in [0..n_c], \\ \ell_k(\tau) &= \prod_{l \in [1..n_c] \setminus \{k\}} \frac{\tau - \tau_l}{\tau_k - \tau_l}, \quad \forall k \in [1..n_c].\end{aligned}$$

Note that the basis polynomials are the same for all elements, due to the normalized time.

In order to obtain the polynomial approximation of the state derivative \dot{x} in element i , the collocation polynomial x_i is differentiated with respect to time. Using (4), (5) and the chain rule, we obtain

$$\begin{aligned}\dot{x}_i(\tau) &= \frac{dx_i}{d\tilde{t}_i}(\tau) = \frac{d\tau}{d\tilde{t}_i} \cdot \frac{dx_i}{d\tau}(\tau) \\ &= \frac{1}{h_i \cdot (t_f - t_0)} \cdot \sum_{k=0}^{n_c} x_{i,k} \cdot \frac{d\tilde{\ell}_k}{d\tau}(\tau).\end{aligned}\quad (6)$$

There are different schemes for choosing the collocation points $\tau_{i,k}$, with different numerical properties, in particular regarding stability and order of convergence. The most common ones are called Gauss, Radau and Lobatto collocation. In this paper we use Radau collocation, which always places a collocation point at the end of each element, and the rest are chosen in a manner that maximizes accuracy.

Collocation methods are not only used for optimization purposes, but are also widely used for numerical solution of both ODE and DAE systems, i.e. simulation. The concepts are the same in both simulation and optimization, and there is a theoretical basis shared by collocation methods in the two areas. See [8] for more on simulation using collocation methods, which are a special case of implicit Runge-Kutta methods.

3.2 Transcription of the dynamic optimization problem

In this section, the infinite-dimensional dynamic optimization problem (1) is transcribed into a finite-dimensional NLP, using the collocation polynomials constructed in the previous section. The main idea is that the infinite-dimensional time-dependent variable z is approximated using polynomials, which can be represented using a finite number of values: the collocation point values. This finite-dimensional approximation of the solution z is more suitable when employing numerical optimization methods.

As decision variables in the NLP we choose all the collocation point values $z_{i,k}$, the state values at the start of each element $x_{i,0}$ and the free parameters p . We also

choose the initial condition values as NLP variables, which we denote by $z_{1,0}$. Finally, we choose t_0 and t_f as optimization variables if they are free. We thus let

$$\begin{aligned}Z = & (z_{1,0}, z_{1,1}, z_{1,2}, \dots, z_{1,n_c}, \\ & x_{2,0}, z_{2,1}, z_{2,2}, \dots, z_{2,n_c}, \\ & x_{3,0}, z_{3,1}, z_{3,2}, \dots, z_{3,n_c}, \\ & \vdots, \\ & x_{n_e,0}, z_{n_e,1}, z_{n_e,2}, \dots, z_{n_e,n_c}, p, t_0, t_f).\end{aligned}$$

be the vector containing all the NLP variables. There are other possibilities in the choice of NLP decision variables, and the choice depends on the collocation scheme. With Radau collocation and the above choice, the transcription of (1) results in the following NLP:

$$\text{min. } \tilde{f}(Z), \quad (7a)$$

$$\text{w.r.t. } Z \in \mathbb{R}^{n_Z},$$

$$\text{s.t. } F(t_{i,k}, z_{i,k}, p) = 0, \quad (7b)$$

$$F_0(z_{1,0}, p) = 0, \quad (7c)$$

$$u_{1,0} - \sum_{k=1}^{n_c} u_{1,k} \cdot \ell_k(0) = 0, \quad (7d)$$

$$z_L \leq z_{i,k} \leq z_U, \quad (7e)$$

$$p_L \leq p \leq p_U, \quad (7f)$$

$$g_e(t_{i,k}, z_{i,k}, p) = 0, \quad (7g)$$

$$g_i(t_{i,k}, z_{i,k}, p) \leq 0, \quad (7h)$$

$$G_e(Z_e) = 0, \quad (7i)$$

$$G_i(Z_i) \leq 0, \quad (7j)$$

$$\forall (i,k) \in \{(1,0)\} \cup ([1..n_e] \times [1..n_c]),$$

$$\dot{x}_{j,l} = \frac{1}{h_j \cdot (t_f - t_0)} \cdot \sum_{m=0}^{n_c} x_{j,m} \cdot \frac{d\tilde{\ell}_m}{d\tau}(\tau_l),$$

$$\forall (j,l) \in [1..n_e] \times [1..n_c], \quad (7k)$$

$$x_{n,n_c} = x_{n+1,0}, \quad \forall n \in [1..n_e - 1], \quad (7l)$$

where

$$n_Z = (1 + n_e \cdot n_c) \cdot n_z + (n_e - 1) \cdot n_x + n_p + 2$$

is the number of scalar NLP variables and

$$t_{i,k} := \tilde{t}_i(\tau_k)$$

denotes collocation point k in element i . The objective (1a) is transcribed into (7a). In the case of optimal control, the Mayer term of the Bolza functional (2) is straightforward to transcribe as

$$\phi(t_0, z(t_0), t_f, z(t_f), p) = \phi(t_0, z_{1,0}, t_f, z_{n_e, n_c}, p).$$

To transcribe the Lagrange term, we start by using (4) to define the Lagrange integrand in element i as

$$L_i(\tau, z_i(\tau), p) := L(\tilde{t}_i(\tau), z(\tilde{t}_i(\tau)), p).$$

The Lagrange term is then approximated as follows.

$$\begin{aligned} & \int_{t_0}^{t_f} L(t, z(t), p) dt \\ &= \sum_{i=1}^{n_e} \left(h_i \cdot (t_f - t_0) \cdot \int_0^1 L_i(\tau, z_i(\tau), p) d\tau \right) \\ &\approx \sum_{i=1}^{n_e} \left(h_i \cdot (t_f - t_0) \cdot \sum_{k=1}^{n_c} \omega_k \cdot L_i(\tau_k, z_{i,k}, p) \right), \end{aligned}$$

where the quadrature weights ω_k are given by

$$\omega_k = \int_0^1 \ell_k(\tau) d\tau.$$

These quadrature weights provides the best approximation for these interpolation points, as shown in [1]. The optimal control objective is thus transcribed as

$$\begin{aligned} f(z, p) &\approx \phi(t_f, z_{n_e, n_c}, p) + \\ & \sum_{i=1}^{n_e} \left(h_i \cdot (t_f - t_0) \cdot \sum_{k=1}^{n_c} \omega_k \cdot L_i(\tau_k, z_{i,k}, p) \right) \\ &=: \tilde{f}(Z). \end{aligned}$$

For the parameter estimation problem, the continuous weighted least squares integral (3) is approximated using the same Gaussian quadrature, resulting in

$$\begin{aligned} f(z, p) &= \int_{t_0}^{t_f} (y(t) - y_m(t)) \cdot Q \cdot (y(t) - y_m(t)) dt \\ &\approx \sum_{i=1}^{n_e} \left(h_i \cdot (t_f - t_0) \cdot \sum_{k=1}^{n_c} \omega_k \cdot (y_{i,k} - y_m(t_{i,k})) \cdot Q \cdot (y_{i,k} - y_m(t_{i,k})) \right) \\ &=: \tilde{f}(Z), \end{aligned}$$

where y_i denotes the collocation polynomials for the measured variables, and $y_{i,k}$ denotes the corresponding collocation point values.

The system dynamics constraint (1b) is only enforced at the collocation points and the start time in the NLP, rather than during the entire time horizon. The initial conditions (1c) are straightforward to transcribe into (7c), since all the initial values have been chosen as NLP variables. The consistency of the user-provided initial conditions is ensured by enforcing all the dynamic constraints at the start time.

The initial values for the states and algebraic variables are determined by the dynamic and initial constraints. The initial value for the control variable is

however not governed by the dynamic or initial equations, but is instead given by the collocation polynomial u_1 . To obtain the value for $u_{1,0}$, we thus need to add the extrapolation constraint (7d).

The bounds and path constraints (1d) to (1g) are straightforward to transcribe into (7e) to (7h), by only enforcing them at the collocation points. How to transcribe the point constraints (1h) and (1i) is less obvious. The approach we have chosen is to assume that each constraint point coincides with a collocation or mesh point. It is then just a matter of identifying the NLP variables that correspond to the constraint point values Z_e and Z_i in order to transcribe the point constraints into (7i) and (7j). The other possibility is to evaluate the collocation polynomials at the constraint points. These constraints are however more computationally expensive to evaluate. But adding elements in order to line up the mesh with the constraint points is prone to be even more expensive. However, as long as the number of constraint points are few in number, which often is the case, this is not a critical issue.

A similar situation occurs during parameter estimation if a discrete least squares sum is used as the objective. The measured variable values are then needed at each of the measurement time points, and these are typically not few in number. The question of whether to line up the mesh (or even collocation) points with the measurement time points, or to simply evaluate the collocation polynomials, is then a critical choice. In this paper however, we avoid this issue by instead using the continuous least squares objective (3). This allows us to evaluate the objective using quadrature, for which we only need the variable values at the collocation points, which are readily available.

In order to determine the state derivative values at the collocation points, we enforce equation (6) at all the collocation points, giving us the collocation equations (7k). These are not enforced at the start time, where the state derivative values instead are determined by the DAE system and initial conditions.

Finally, we add the continuity constraints (7l), to get continuity for the state. An NLP has the general form

$$\begin{aligned} & \text{minimize} && f(x) \\ & \text{with respect to} && x \in \mathbb{R}^{n_x}, \\ & \text{subject to} && x_L \leq x \leq x_U, \\ & && g(x) = 0, \\ & && h(x) \leq 0, \end{aligned}$$

which the transcription (7) is a special case of. By solving the NLP (7), we may obtain an approximate solution to the dynamic optimization problem (1).

4 Tools

4.1 CasADi

Obtaining the first and second-order derivatives of the NLP cost and constraints functions with respect to the NLP variables allows for efficient solution of an NLP. To this end, CasADi [9] (Computer algebra system with Automatic Differentiation) is used. CasADi is a low-level tool for efficiently computing derivatives using automatic differentiation (AD) and is tailored for dynamic optimization. Once a symbolic representation of an NLP has been created using CasADi, the needed derivatives are efficiently and conveniently obtained and sparsity patterns are preserved.

To solve the NLP (7), we use IPOPT [10]. IPOPT uses a sparse primal-dual interior point method to find local optima to large-scale NLPs. CasADi comes with an interface to IPOPT, which is used in the implementation.

4.2 JModelica.org

4.2.1 The JModelica.org platform

JModelica.org [3] is an open-source platform for simulation and optimization of Modelica models. Whereas standard Modelica tools, such as Dymola¹ and OpenModelica², mostly focus on the simulation of physical systems, JModelica.org also targets large-scale dynamic optimization. A common problem is that a large amount of research goes into developing algorithms without accompanying means of describing complex physical systems, making these algorithms difficult to use in practical applications. One goal of JModelica.org is to open up the Modelica language and the vast amount of existing Modelica models to algorithms developed in academia.

The Modelica language is largely designed with simulation-based analysis in mind. To accommodate the need for conveniently formulating dynamic optimization problems based on models described by Modelica code, the Modelica extension Optimica [4] has been developed and integrated into JModelica.org. Optimica enables the extension of a Modelica model to include the constructs used to formulate a dynamic optimization problem, such as (1), where the pure Modelica code describes the dynamic constraints (1b) and (1c).

¹<http://www.3ds.com/products/catia/portfolio/dymola>

²<http://www.openmodelica.org/>

The main components of JModelica.org are the Modelica and Optimica compilers, which are implemented in Java, and the three modeling interfaces Functional Mock-up Interface (FMI)³, JModelica.org Model Interface (JMI) and a new symbolic XML-based format based on the FMI XML format, which includes equations in symbolic form. The user interacts with the various components of JModelica.org via the scripting language Python.

While FMI is a standard defining a tool-independent format for representation of hybrid dynamic models on ordinary differential equation (ODE) form, JMI is a runtime library designed solely for JModelica.org, and has long been the main interface for dynamic optimization in JModelica.org. The main optimization algorithm in JMI is collocation-based and implemented in C. It relies on CppAD⁴ to compute and evaluate derivatives. However, in this paper the new XML-based format is instead used for the new collocation algorithm. This format is an extension of the XML format used in FMI and is described in [11]. The format uses a DAE representation of the model instead of an ODE representation. It is designed to use a model representation that is as general as possible, allowing for the formulation of a wide variety of problems based on Modelica code, in particular dynamic optimization problems described by Optimica code. CasADi supports import of models described by this XML format, allowing for smooth interaction between JModelica.org and CasADi.

4.2.2 The collocation algorithm toolchain

Figure 1 depicts an overview of the entire workflow for the implemented collocation algorithm.

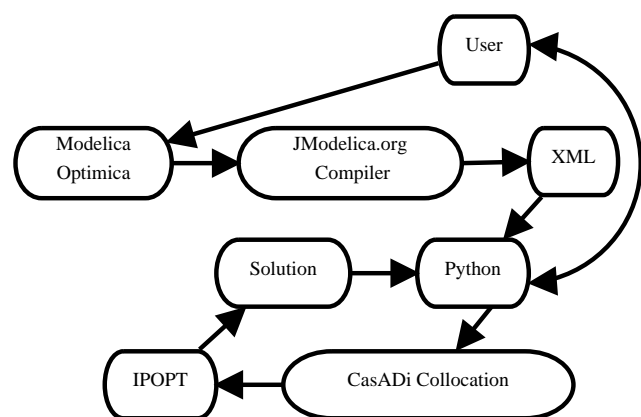


Figure 1: Overview of algorithm workflow

³<http://www.functional-mockup-interface.org/>

⁴<http://www.coin-or.org/CppAD/>

The user starts by defining the system model in Modelica and the dynamic optimization problem in Optimica. The user interaction is carried out in Python. The Optimica file is, via Python, sent to JModelica.org's compiler. The compiler generates an XML file from the Optimica file, which has a flat, rather than hierarchical, representation of the dynamic optimization problem similar to that of (1).

The XML file is parsed by CasADi and JModelica.org and the extracted information is used to transcribe the problem into an NLP by the collocation algorithm inside JModelica.org. This NLP problem is then solved by IPOPT. The solution is written to a result file in a format compliant with Dymola. The solution is also represented by a Python object which is returned to the user. This allows the user to freely analyze the data in Python, e.g. plotting it either manually or using JModelica.org's plotting GUI.

5 Implementation extensions

The work presented in this paper is a continuation of the work begun in [5], where a prototypical collocation algorithm was implemented in JModelica.org using CasADi in Python. In this section we describe the prominent extensions made to this implementation.

The algorithm supports problems with free start and final time. Since these are typically combined with terminal constraints, support for general point constraints has also been added.

Whereas the old implementation only supported Radau collocation with three collocation points per element, the new implementation supports an arbitrary number of collocation points (up to about 80 points, at which point the method for computing the collocation points runs into numerical problems). The new implementation also supports Gauss collocation as an alternative to Radau collocation.

CasADi has two separate approaches to performing automatic differentiation. The first approach is called SX and is a conventional AD approach, where the computation graph is only allowed to contain scalar, built-in unary and binary operations. The second approach is called MX and allows for more general operations in the computation graph, such as matrix operations (preserving sparsity), branches and user-defined functions. The novel MX graphs are less computationally efficient than SX graphs, but support a wider range of operations. This allows the resulting MX graphs to be smaller than SX graphs, thus consuming less memory, which may be critical. The collocation algorithm

has been extended to enable the user to choose between SX and MX graphs. See [9] for more details regarding SX and MX graphs.

The collocation algorithm only deals with systems which are continuous in time. However, control signals are often inherently discrete in time, which can not be disregarded in for example model predictive control. In order to support discrete control signals, the possibility of adding blocking factors has been added. Blocking factors change the representation of control signals from piecewise polynomial to piecewise constant. Control signals can be forced to remain constant over single or multiple elements.

Finally, options have been added to allow the elimination of certain NLP variables. The state derivative variables $\dot{x}_{i,k}$ can be eliminated by inlining the collocation constraint (7k), and the state continuity variables $x_{i,0}$ can be eliminated by inlining the continuity constraint (7l). This allows for the trade-off between problem size and problem sparsity. Eliminating state derivatives also has the benefit of no longer needing to scale these variables, which often is difficult.

6 Benchmarks

6.1 Benchmark setting

In this section, we will compare the newly extended collocation algorithm based on CasADi and its Python interface with the old collocation algorithm implemented in C. Both of these algorithms are implemented in JModelica.org. We use Radau collocation with the same, low number of collocation points per element. The benchmarks are based on a continuously stirred tank reactor and a combined cycle power plant.

The two algorithms are based on the same theory and the constructed NLP problems are nearly identical, so the solutions can be expected to also be nearly identical. There are however a few differences. The most prevalent is that the new algorithm constructs AD graphs for the entire NLP. The computation of the Hessian of the Lagrangian function is thus easy and efficient. Obtaining this information for the old algorithm using CppAD, although possible, would require a tremendous effort to implement, which has not been done. Thus IPOPT employs a quasi-Newton method for the old algorithm, in which the Hessian instead is approximated. The computation of the Hessian and AD graphs for the entire NLP is expensive. These computations can however be performed off-line, and in turn make the on-line computations more effective.

In this benchmark, SX graphs are used for the new algorithm, since the generality offered by MX graphs are unnecessary for the presented benchmarks.

All the benchmarks are run on a Fedora 16 computer with an Intel® Core™ i7-2600 Quad processor @ 3.4 GHz. Revisions [3352] and [2594] of JModelica.org and of CasADi respectively are used, together with version 3.10.2 of IPOPT with the MA27 linear solver. For each benchmark, we provide the following run-time statistics:

- Off-line: The CPU time [s] spent doing off-line computations, which includes compilation of the Modelica and Optimica code, construction of AD graphs and computation of derivatives of NLP functions.
- On-line: The CPU time [s] spent doing on-line computations, which essentially is the time spent in IPOPT. This part consists of two parts, where the first one is the time spent internally in IPOPT, and the second part is the time spent evaluating NLP functions, which is done by CppAD or CasADi. The time spent by CasADi evaluating NLP functions is nearly negligible, whereas CppAD spends a significant amount of time evaluating functions on-line for the old algorithm.
- Total: The total CPU time [s] from the start of the compilation until the optimization result is returned.
- Iterations: The number of iterations required by IPOPT to solve the problem.

Minor variations in the collocation scheme or problem formulation can have a tremendous impact on the required number of iterations, for example if the solver has to enter a restoration phase, which in turn affect the overall solution time. But on average, the required number of iterations for the two algorithms should be similar for a specific problem. The only significant reason to expect a different number of iterations is due to that the new algorithm computes second-order derivatives analytically, whereas the old algorithm approximates them numerically. The number of iterations for the new algorithm can thus be expected to be lower on average.

6.2 Continuously stirred tank reactor

The continuously stirred tank reactor (CSTR) model used for this benchmark was developed in [12]. The

system contains a highly nonlinear exothermic reaction and has two states: reactant concentration c [mol/m³] and reactor temperature T [K]. The rate F_0 [m³/s], concentration c_0 [mol/m³] and temperature T_0 [K] of the reactant inflow are assumed to be constant. The reactor has a liquid cooling system, whose temperature T_c [K] is the sole control variable.

A formulation analogous to (1) of the considered problem is to

$$\min. \quad \phi(t_f), \quad (8a)$$

$$\text{w.r.t.} \quad c, T, T_c, \phi,$$

$$\text{s.t.} \quad \dot{c}(t) = F_0 \cdot \frac{c_0 - c(t)}{V} - k_0 \cdot e^{-\frac{E_a}{T(t)}} \cdot c(t), \quad (8b)$$

$$\begin{aligned} \dot{T}(t) = & F_0 \cdot \frac{T_0 - T(t)}{V} - \\ & \frac{H}{\rho \cdot C_p} \cdot k_0 \cdot e^{-\frac{E_a}{T(t)}} \cdot c(t) + \\ & \frac{2 \cdot U}{r \cdot \rho \cdot C_p} \cdot (T_c(t) - T(t)), \end{aligned} \quad (8c)$$

$$\dot{\phi}(t) = \|(c(t), T(t), T_c(t)) - (c^{\text{ref}}, T^{\text{ref}}, T_c^{\text{ref}})\|_2^2 \quad (8d)$$

$$(c(t_0), T(t_0), T_c(t_0), \phi(t_0)) = (c_0, T_0, T_{c0}, 0), \quad (8e)$$

$$(T(t), T_c(t)) \leq (350, 370), \quad (8f)$$

$$\forall t \in [t_0, t_f].$$

The objective (8a) is to move the system from the stationary operation point given by the initial condition (8e), where

$$(c_0, T_0, T_{c0}) \approx (956.3, 250.1, 370.0),$$

to the stationary operation point, given by

$$(c^{\text{ref}}, T^{\text{ref}}, T_c^{\text{ref}}) \approx (338.8, 280.1, 280.0).$$

The variable ϕ is introduced as a state and measures how the cost increases over time, and is governed by the dynamic equation (8d). This allows us to formulate the objective on Mayer form, instead of Lagrange.

The dynamics of the system are modelled by equations (8b) and (8c), where $V, k_0, E_a, H, \rho, C_p, U$ and r are physical parameters and constants. In order to avoid too high temperatures, we impose the variable bounds (8f). With $t_f = 200$ s, $n_e = 70$ and $n_c = 5$, we get the following result.

Table 1: Run-time statistics for the CSTR benchmark

	Off-line	On-line	Total	Iterations
New alg.	1.0	0.3	1.3	50
Old alg.	2.0	0.9	2.9	62

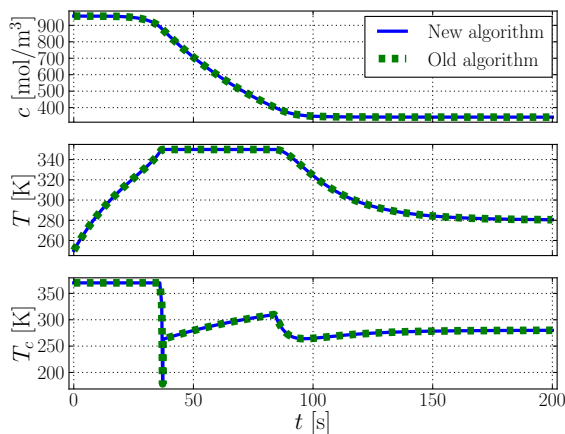


Figure 2: Comparison of the old and new algorithm on optimal control of a CSTR

We see that for this benchmark, the new algorithm is about twice as fast both off-line and on-line. They also produce the same solution (up to IPOPT tolerances). This problem is very small-scale, and in the next benchmark we will see that a larger problem will allow the new algorithm to truly outperform the old one

6.3 Combined cycle power plant

The combined cycle power plant (CCPP) model used for this benchmark is described in [13]. The model has 9 states, 128 algebraic variables and 1 control variable. The task is to minimize the time required to perform a warm start-up of the power plant. This problem has become highly industrially relevant during the last few years, due to an increasing need to improve power generation flexibility. The startup process is considered finished when the normalized load input signal u [1] to the steam turbine, starting at 15 %, has reached 100 % and the evaporator pressure p [Pa], which is a state with an initial value of approximately 3.47 MPa, has reached approximately 8.35 MPa.

In order to reduce the wear and tear on the steam turbine, which is one of the most expensive parts of the power plant, the thermal stress in the turbine σ [Pa], which is an algebraic variable, may not exceed 260 MPa. This is the main limiting factor in the startup process. Another imposed constraint is that the derivative of the load input signal u may not be negative and may not exceed $0.1/60 \text{ s}^{-1}$. Since these bounds are on the derivative of the control variable, which is not supported by neither the old nor the new algorithm, we

introduce the control variable \dot{u} and add the equation

$$\frac{du}{dt} = \dot{u},$$

to the DAE system. This converts the previous control variable u into a state, giving us a total of 10 states, and the sole control variable is now instead \dot{u} , which we can impose the mentioned bounds on.

We formulate a Lagrange cost function which penalizes the weighted deviation of the load input signal and the evaporator pressure from their respectively desired values, given by

$$f(z) = \int_{t_0}^{t_f} \left(10^{-12} \cdot (p(t) - 8.35 \cdot 10^6)^2 + 0.5 \cdot (u(t) - 1)^2 \right) dt.$$

With $t_0 = 0 \text{ s}$, $t_f = 4000 \text{ s}$, $n_e = 40$ and $n_c = 4$, the following optimization result is obtained.

Table 2: Run-time statistics for the CCPP benchmark

	Off-line	On-line	Total	Iterations
New alg.	4.9	3.0	7.9	79
Old alg.	13.2	23.9	37.2	75

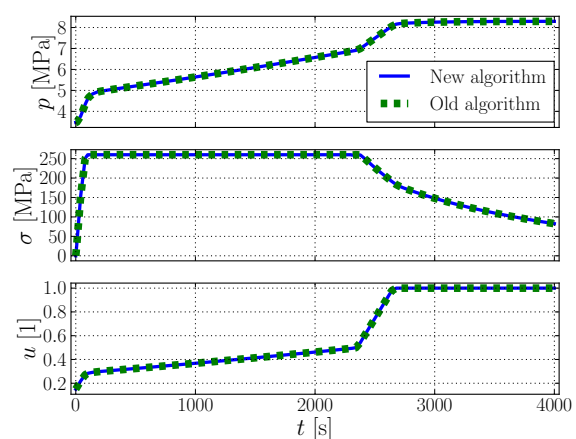


Figure 3: Comparison of the old and new algorithm for optimal start-up of a CCPP

In this case we clearly see the benefits of constructing AD graphs for the entire NLP problem using CasADi for large-scale problems, which is what allows for the exceptionally quick NLP on-line solution. Once again the algorithms find the same solution.

7 Conclusions

We have presented and implemented an optimization algorithm based on existing theory for direct collocation. The algorithm has been compared to an old and similar algorithm in JModelica.org. The solutions found by the two algorithms have shown to be as identical as can be expected, that is, up to IPOPT tolerances.

The overall performance of the new algorithm compared to the old algorithm, in terms of speed, is clearly superior, especially for large-scale problems. In terms of being fully-featured, there are still a few important features missing for the new algorithm. CasADi combined with Python is however very flexible, so adding new features is often straightforward, which is not the case for the old algorithm implemented in C.

Future work includes adding additional discretization schemes, adding support for multi-phase problems and allowing element lengths to be free in order to maximize accuracy. A related topic is the further development of Optimica, to support additional problem formulations.

References

- [1] L. T. Biegler, *Nonlinear Programming: Concepts, Algorithms, and Applications to Chemical Processes*. MOS-SIAM Series on Optimization, Mathematical Optimization Society and the Society for Industrial and Applied Mathematics, 2010.
- [2] T. Binder, L. Blank, H. Bock, R. Bulirsch, W. Dahmen, M. Diehl, T. Kronseder, W. Marquardt, J. Schlöder, and O. Stryk, "Introduction to model based optimization of chemical processes on moving horizons," in *Online Optimization of Large Scale Systems: State of the Art* (M. Grötschel, S. Krumke, and J. Rambau, eds.), pp. 295–340, Springer, 2001.
- [3] J. Åkesson, K.-E. Årzén, M. Gäfvert, T. Bergdahl, and H. Tummescheit, "Modeling and optimization with Optimica and JModelica.org—languages and tools for solving large-scale dynamic optimization problem," *Computers and Chemical Engineering*, vol. 34, pp. 1737–1749, Nov. 2010.
- [4] J. Åkesson, "Optimica—an extension of Modelica supporting dynamic optimization," in *In 6th International Modelica Conference 2008*, Modelica Association, Mar. 2008.
- [5] J. Andersson, J. Åkesson, F. Casella, and M. Diehl, "Integration of CasADi and JModelica.org," in *8th International Modelica Conference*, Mar. 2011.
- [6] F. Magnusson, "Collocation methods in JModelica.org," Master's Thesis ISRN LUTFD2/TFRT-5892--SE, Feb. 2012.
- [7] J. T. Betts, *Practical Methods for Optimal Control and Estimation using Nonlinear Programming*. SIAM's Advances in Design and Control, Society for Industrial and Applied Mathematics, 2nd ed., 2010.
- [8] E. Hairer and G. Wanner, *Solving Ordinary Differential Equations II: Stiff and differential-algebraic problems*. Springer series in computational mathematics, Springer-Verlag, 2nd ed., 1996.
- [9] J. Andersson, J. Åkesson, and M. Diehl, "CasADi – A symbolic package for automatic differentiation and optimal control," in *Recent Advances in Algorithmic Differentiation* (S. Forth, P. Hovland, E. Phipps, J. Utke, and A. Walther, eds.), Lecture Notes in Computational Science and Engineering, (Berlin), Springer, 2012.
- [10] A. Wächter and L. T. Biegler, "On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming," *Mathematical Programming*, vol. 106, no. 1, pp. 25–57, 2006.
- [11] R. Parrotto, J. Åkesson, and F. Casella, "An XML representation of DAE systems obtained from continuous-time Modelica models," in *3rd International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools*, (Oslo, Norway), pp. 91–98, Oct. 3 2010.
- [12] G. A. Hicks and W. H. Ray, "Approximation methods for optimal control synthesis," *The Canadian Journal of Chemical Engineering*, vol. 49, no. 4, pp. 522–528, 1971.
- [13] F. Casella, F. Donida, and J. Åkesson, "Object-oriented modeling and optimal control: A case study in power plant start-up," in *18th IFAC World Congress*, (Milano, Italy), Aug. 2011.

Parallel Multiple-Shooting and Collocation Optimization with OpenModelica

Bernhard Bachmann¹, Lennart Ochel¹, Vitalij Ruge¹,
Mahder Gebremedhin², Peter Fritzson²,
Vaheed Nezhadali³, Lars Eriksson³, Martin Sivertsson³

¹Dept. Mathematics and Engineering, University of Applied Sciences, D-33609 Bielefeld, Germany

²PELAB – Programming Environment Lab, Dept. Computer Science
Linköping University, SE-581 83 Linköping, Sweden

³Vehicular Systems, Dept. Electrical Engineering
Linköping University, SE-581 83 Linköping, Sweden

{bernhard.bachmann,lennart.ochel,vitalij.ruge}@fh-bielefeld.de,
{peter.fritzson,mahder.gebremedhin,vaheed.nezhadali,lars.eriksson,marsi}@liu.se

Abstract

Nonlinear model predictive control (NMPC) has become increasingly important for today's control engineers during the last decade. In order to apply NMPC a nonlinear optimal control problem (NOCP) must be solved which in general needs high computational effort.

State-of-the-art solution algorithms are based on multiple shooting or collocation algorithms, which are required to solve the underlying dynamic model formulation. This paper describes a general discretization scheme applied to the dynamic model description which can be further concretized to reproduce the multiple shooting or collocation approach. Furthermore, this approach can be refined to represent a total collocation method in order to solve the underlying NOCP much more efficiently. Further speedup of optimization has been achieved by parallelizing the calculation of model specific parts (e.g. constraints, Jacobians, etc.) and is presented in the coming sections.

The corresponding discretized optimization problem has been solved by the interior optimizer Ipopt. The proposed parallelized algorithms have been tested on different applications. As industrial relevant application an optimal control of a Diesel-Electric power train has been investigated. The modeling and problem description has been done in Optimica and Modelica. The simulation has been performed using OpenModelica. Speedup curves for parallel execution are presented.

Keywords: Modelica, Optimica, optimization, multiple shooting, collocation, parallel, simulation

1 Introduction

This paper presents efficient parallel implementations and measurement results of solution methods for nonlinear optimal control problems (NOCP) relevant for nonlinear model predictive control (NMPC) applications.

NMPC as well as NOCP have become increasingly important for industrial applications during the last decade [3], [4]. State-of-the-art solution algorithms [4] are based on multiple shooting or collocation algorithms, which are needed to solve the underlying dynamic model formulation. This paper concentrates on parallelizing these time-consuming algorithms, which finally lead to a very fast solution of the underlying NOCP. Moreover, a general discretization scheme applied to the dynamic model description is introduced, which can be further concretized to reproduce the common multiple shooting or collocation approach [7] and can also be refined to represent total collocation methods [4] in order to solve the underlying NOCP much more efficiently. The modeling and problem description is done in Modelica [2] extended with optimization goal functions and constraints specified as in Optimica [15]. The simulation is performed using OpenModelica [1]. Speedup curves for parallel execution are presented for application examples.

Section 2 describes the underlying mathematical problem formulation including the objective function and constraints to the state and control variables. The general discretization scheme applied is discussed in Section 3. This approach can be further refined to represent multiple shooting or collocation algorithms for the solution process, which is described in Section 4.

In section 5 the general discretization scheme is further developed towards total collocation methods.

Industrial relevant Modelica applications are presented in Section 6. Parallel execution of the constraint equations of the NOCP is performed in Section 7. The results show reasonable speedups of the optimization time when it comes to time consuming calculation of the model equations. The necessary implementations are partly realized in the OpenModelica Compiler, which is described in Section 8. The paper concludes with a summary of the achieved results.

2 The Nonlinear Optimal Control Problem (NOCP)

The numerical solution of NOCP is performed by solving the following problem formulation [7][8]:

$$\begin{aligned} \min_{u(t)} J(x(t), u(t), t) \\ = E(x(t_f)) \\ + \int_{t_0}^{t_f} L(x(t), u(t), t) dt \end{aligned} \tag{2.1}$$

subject to

$$\begin{aligned} x(t_0) &= h_0 & (2.2) \\ \dot{x}(t) &= f(x(t), u(t), t) & (2.3) \\ g(x(t), u(t), t) &\geq 0 & (2.4) \\ r(x(t_f)) &= 0 & (2.5) \end{aligned}$$

where $x(t) \in \mathbb{R}^{\eta_x}$ and $u(t) \in \mathbb{R}^{\eta_u}$ are the state and control variables, respectively. The receding time horizon is given by the interval $[t_0, t_f]$. The constraints (2.2), (2.3), (2.4) and (2.5) describe the initial conditions, the nonlinear dynamic model description based on differential algebraic equations (DAEs, Modelica), the path constraints ($g(x(t), u(t), t) \in \mathbb{R}^v$) and the terminal constraints.

Support for time-optimal control and corresponding terminal constraints is work-in-progress and are not yet provided by the current implementation.

2.1 Boundary Value Problems

The objective function (2.1), that needs to be minimized, includes conditions at the boundary time point t_f stated by the function $E(x(t_f))$ as well as conditions taking into account the whole time horizon stated by $\int_{t_0}^{t_f} L(x(t), u(t), t) dt$.

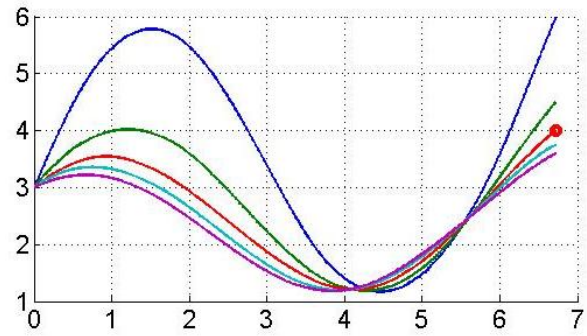


Figure 1. Different trajectories achieved by varying control variables. Only one trajectory fulfills the terminal constraint (red dot).

The function $E(x(t_f))$ describes conditions that should be fulfilled at the final time point similar to the terminal constraint (2.5). Since $E(x(t_f))$ is part of the objective function $J(x(t), u(t), t)$ the applied optimization methods may not find a solution that fulfills the corresponding terminal constraints, but should be very close to it. The trajectories are influenced by changing the control variables. Different trajectories using different control variables are visualized in Figure 1.

On the other hand, different trajectories could fulfill the same terminal constraints. Taking into account the whole time horizon by minimizing the second part $\int_{t_0}^{t_f} L(x(t), u(t), t) dt$ of the objective function will lead to the selection of the optimal trajectory. This behavior is visualized in Figure 2.

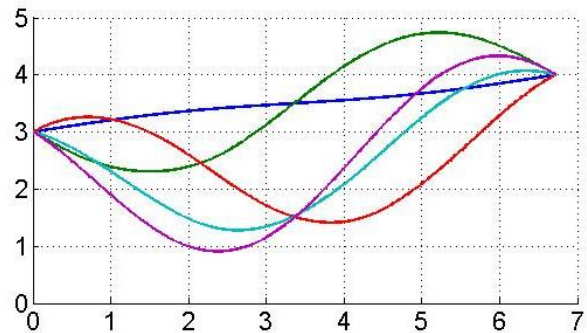


Figure 2. Different trajectories that fulfill the terminal constraint.

3 General Discretization Scheme

In order to apply a general discretization scheme the NOCP formulation is rewritten to a general form which later can be used to derive the different possible numerical algorithms e.g. multiple shooting, multiple or total collocation algorithm, etc. [6]. Equations (2.2) and (2.3) can be rewritten as follows:

$$x(\tau) = h_0 + \int_{t_0}^{\tau} f(x(t), u(t), t) dt. \tag{3.1}$$

When discretizing the time horizon $[t_0, t_f]$ into a finite number of intervals $[t_0, t_1], \dots, [t_{n-1}, t_n]$ (e.g. equidistant partitioning: $t_j := t_0 + l \cdot j$, $j = 0, \dots, n$, $l := \frac{t_f - t_0}{n}$) integral in (3.1) can be reformulated to

$$\int_{t_0}^{\tau} f(x(t), u(t), t) dt = \sum_{i=0}^{n-1} \int_{t_i}^{t_{i+1}} f(x(t), u(t), t) dt. \quad (3.2)$$

Each integral

$$\int_{t_i}^{t_{i+1}} f(x(t), u(t), t) dt \quad (3.3)$$

on a subinterval can now be treated independently, if additional constraints are added to the NOCP formulation to force the calculation of an overall continuous solution. Therefore, locally the problem reduces to a boundary value problem [5] stated by

$$x_i(t_{i+1}) = h_i + \int_{t_i}^{t_{i+1}} f(x_i(t), u(t), t) dt \quad (3.4)$$

where $x_i(t) := x(t)$ for $t \in [t_i, t_{i+1}]$, $i = 0, \dots, n-1$. It yields $x_i(t_i) = h_i$ and continuity is forced by additional constraints $x_i(t_{i+1}) = h_{i+1}$ added to the NOCP formulation, which finally leads locally to a boundary value problem. Each sub-problem (3.4) can be solved independently and in parallel, if multiple shooting/collocation is applied. By varying the control variable $u(t)$ in each sub-interval the solution of (3.4) can be influenced in order to fulfill the overall continuity constraints. In the current approach it is assumed that $u(t) = u_i$ is constant for each subinterval $[t_i, t_{i+1}]$.

4 Multiple Shooting or Collocation

Different numerical methods are available to solve equation (3.4). The first approach presented within this paper is the reformulation of (3.4) to an ordinary differential equation

$$\dot{x}_i(t) = f(x_i(t), u_i, t) \quad (4.1)$$

with the initial condition $x_i(t_i) = h_i$.

In order to solve equation (4.1) an appropriate (e.g. explicit/implicit) integration algorithm can be applied that is already available in OpenModelica. A schematic view of the algorithmic dependencies is presented in **Figure 3**.

Alternatively, equation (3.4) or (4.1) can locally be solved using collocation methods, which also can be interpreted as numerical treatment of integration. De-

tailed descriptions of the multiple shooting algorithm using local collocation can be found in [7]. The solution process for equation (3.4) in each subinterval can be performed in parallel. The necessary calculation time depends certainly on the chosen integration method. In case of an explicit integration algorithm, e.g. Runge-Kutta based, more intermediate integration steps might be necessary for certain accuracy than using an implicit integration method, e.g. local collocation methods. On the other hand, explicit integration methods just perform at each intermediate step an evaluation of the model equations, whereas implicit methods in general need to solve a system of non-linear equations, which might also be time consuming. Nevertheless, when the underlying system of ordinary differential equations is stiff, implicit methods need to be applied.

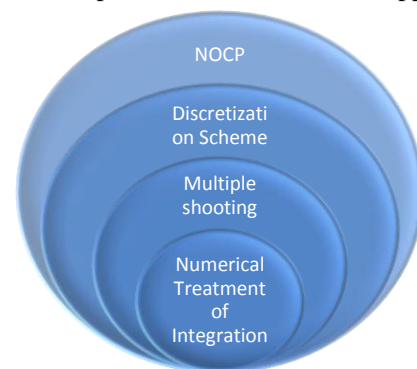


Figure 3. Schematic view of the algorithmic dependencies.

Although, equation (3.4) can be solved in parallel a lot of time is used for finding exact solutions to a locally defined problem, which might not be relevant for the over-all problem stated by the (NOCP) formulation (2.1)-(2.5). Therefore, the solution process for the NOCP still needs a lot of computation time. The next section describes methods to overcome this deficiency by adding the locally derived residual equations (based on locally applied collocation methods) to the over-all NOCP formulation.

5 Total Collocation

Applying collocation methods for solving equation (3.4) locally leads in general to a system of non-linear equations for each sub-interval. The solution process of these equations might be time consuming and with respect to the NOCP not efficient. If the corresponding non-linear equations are added to the NOCP formulation and corresponding optimization algorithms have access to the intermediate points used by the local collocation method a more efficient solution process can be formulated [4]. This section presents two different collocation methods.

Based on the common Lagrangian polynomial $p_j(z)$ for interpolation purposes, following abbreviations are introduced for $j = 0, \dots, m$ and $k = 1, \dots, m$:

$$p_{k,j} := p_j(z_k) := \prod_{\substack{l=0 \\ l \neq j}}^m \frac{z_k - z_l}{z_j - z_l},$$

$$\frac{\partial p_{k,j}}{\partial z}(z_k) := \sum_{\substack{l1=0 \\ l1 \neq j}}^m \frac{1}{z_{l1} - z_j} \cdot \prod_{\substack{l2=0 \\ l2 \neq j \\ l2 \neq l1}}^m \frac{z_k - z_{l2}}{z_j - z_{l2}} \quad \text{and}$$

$$\int p_{k,j} := \int_0^{z_k} p_j(z) dz$$

where z_0, \dots, z_m are the supporting points within the reference interval $[0,1]$. Further abbreviations are defined by $t_{j,i} = t_i + l \cdot z_j$, $x_{m,0} := h_0$, $x_{j,i} := x_i(t_{j,i})$, and $f_{j,i} := f(x_{j,i}, u_i, t_{j,i})$.

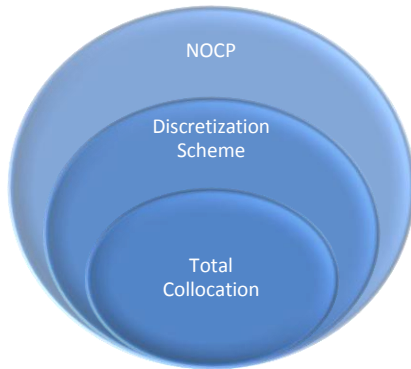


Figure 4. Schematic view of the algorithmic dependencies.

The first variant is dealing with the approximation of the states which leads to the following formulas:

$$x_{k,i} = p_{k,0} \cdot x_{m,i-1} + \sum_{j=1}^m p_{k,j} \cdot x_{j,i} \quad (5.1)$$

$$l \cdot f_{k,i} \approx \phi_{k,i} := \frac{\partial p_{k,0}}{\partial z} \cdot x_{m,i-1} + \sum_{j=1}^m \frac{\partial p_{k,j}}{\partial z} \cdot x_{j,i}$$

In case of $m = 1$ this approach reduces to the implicit Euler formula with approximation order 1.

The second variant is dealing with the approximation of the derivatives of the states and leads to the formulas:

$$x_{k,i} \approx \psi_{k,i} := x_{m,i-1} + l \cdot \sum_{j=0}^m \int p_{k,j} \cdot f_{j,i} \quad (5.2)$$

$$f_{k,i} = \sum_{j=0}^m p_{k,j} \cdot f_{j,i}$$

In case of $m = 1$ this approach reduces to an implicit Runge-Kutta formula (trapezoidal rule) with approximation order 2.

The discretized NOCP using total collocation and corresponding Gaussian quadrature formula for the integral part of the goal function is finally described by:

$$\min_{u(t)} J(x(t), u(t), t) = E(x_{m,n}) + l \cdot \sum_{i=1}^n \sum_{j=0}^m \omega_j \cdot L(x_{j,i}, u_i, t_{j,i}) \quad (5.3)$$

subject to

$$\begin{aligned} s(x_{k,i}, u_i, t_{k,i}) &= 0 \\ u(t_{k,i}) &= u_i \\ g(x_{k,i}, u_i, t_{k,i}) &\geq 0 \\ r(x_{m,n}) &= 0 \end{aligned} \quad (5.4)$$

for $i = 1, \dots, n$, $k = 1, \dots, m$. For variant 1 the supporting points z_1, \dots, z_m , and weights $\omega_1, \dots, \omega_m$ are given based on Radau formulas. $z_0 = 0$, $\omega_0 = 0$. $s(x_{k,i}, u_i, t_{k,i}) = l \cdot f_{k,i} - \phi_{k,i}$ are the additional residual equations from (5.1). For variant 1 the supporting points z_0, \dots, z_m , and weights $\omega_0, \dots, \omega_m$ are given based on Lobatto formulas. $s(x_{k,i}, u_i, t_{k,i}) = x_{k,i} - \psi_{k,i}$ are the additional residual equations from (5.2).

6 Modelica Applications

To investigate the performance of the proposed optimization algorithm, industrial relevant optimal control problems are solved and corresponding results are presented in this section.

6.1 Batch Reactor

We begin by considering a simple model from the chemical reactor described in [7] to maximize the yield of $x_2(t)$ by manipulation the reaction temperature $u(t)$, with the following problem formulation:

$$\min_{u(t)} J(x(t), u(t), t) = -x_2(1) \quad (6.1)$$

subject to

$$\begin{aligned} \dot{x}_1(t) &= -\left(u(t) + \frac{u^2(t)}{2}\right) \cdot x_1(t) \\ \dot{x}_2(t) &= u(t) \cdot x_1(t) \end{aligned} \quad (6.2)$$

$$\begin{pmatrix} x_1(t) \\ 1 - x_1(t) \\ x_2(t) \\ 1 - x_2(t) \\ u(t) \\ 5 - u(t) \end{pmatrix} \geq 0 \quad (6.3)$$

$$x(0) = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad (6.4)$$

where $x(t) = (x_1(t), x_2(t))^T$ and $t \in [0,1]$.

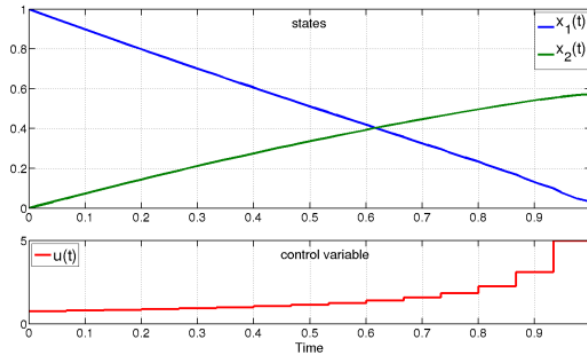


Figure 5. Trajectories of state and control variables

6.2 Optimal control of Diesel-Electric powertrain

The Diesel-electric model based on [10] is presented in Appendix A. This concept is modeled according to a nonlinear mean value engine model (MVEM) containing four states and three control inputs while the generator model is simplified by considering constant efficiency and maximum power over the entire speed range.

In a Diesel-electric powertrain the operating point of the Diesel engine can be freely chosen which would potentially decrease fuel consumption. Moreover, the electric machine has better torque characteristics. These are the main reasons making the Diesel-electric powertrain concept interesting for further studies.

To investigate the fuel optimal transients of the powertrain from idling condition to a certain power level while the accelerator pedal position is interpreted as a power level request, the following optimal control problem is solved:

$$\text{states } x = \begin{pmatrix} \omega_{ice} \\ p_{im} \\ p_{em} \\ \omega_{tc} \end{pmatrix}, \text{ controls } u = \begin{pmatrix} u_f \\ u_{wg} \\ P_{gen} \end{pmatrix}$$

$$\min \int_0^T \dot{m}_f dt$$

subject to

$$\begin{aligned} \dot{x}_1 &= f_2(x_2, x_3, u_1, u_3) \\ \dot{x}_2 &= f_3(x_1, x_2, x_4) \\ \dot{x}_3 &= f_4(x_1, x_2, x_3, u_1, u_2) \\ \dot{x}_4 &= f_5(x_2, x_3, x_4, u_2) \\ 0 &= f_6(x_2, x_4) - f_7(x_1, x_2) \\ 0 &= f_7(x_1, x_2) + f_8(x_1, u_1) - f_9(x_3) - f_{10}(x_3, u_3) \\ 0 &= \frac{f_{11}(x_3) - f_{12}(x_1)}{f_{13}(x_4)} - f_{14}(x_4) \end{aligned}$$

$$\begin{aligned} 54 \text{ rps} &\leq x_1 \leq 220 \text{ rps} \\ 0.8 P_{amb} &\leq x_2 \leq 2P_{amb} \\ P_{amb} &\leq x_3 \leq 3P_{amb} \\ 300 \text{ rps} &\leq x_4 \leq 10000 \text{ rps} \\ 0 &\leq u_1, u_2 \leq 1 \end{aligned}$$

and boundary conditions are:

$$\text{at } t = 0, \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \text{idle operating values,}$$

$$\text{at } t = T, \begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{pmatrix} = 0, \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \text{desired values,}$$

$$\text{and } u_3 = P_{\text{required}}.$$

The constraints are originated from components' limitations and the functions f_i are described in the appendix [10].

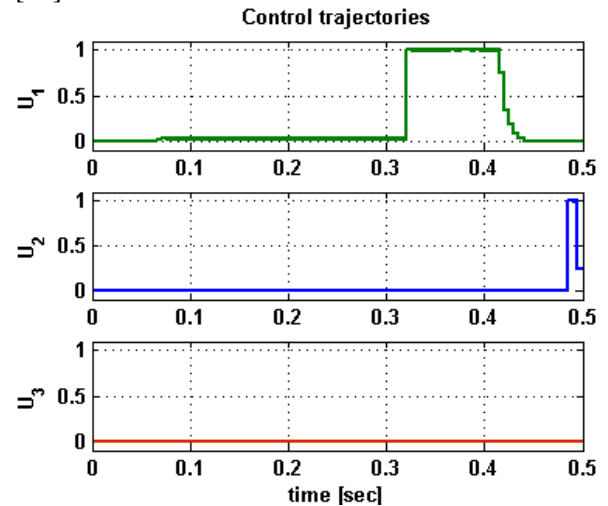


Figure 6. Trajectories of control variables

In this work, we try to find the fuel optimal control and state trajectories in a certain time interval $[0, 0.5]$. For simplicity, only diesel operating condition is assumed which means $(u_3 = P_{gen} = 0)$.

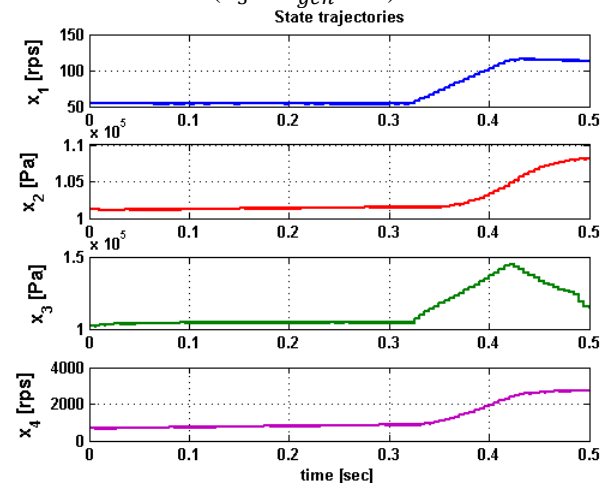


Figure 7. Trajectories of state variables

The dynamic system is solved after it is discretized into subintervals. **Figure 6** and **Figure 7** show the obtained control and state trajectories. As it is expected, the fuel optimal results happen when engine is accelerated only near the end of the time interval ($t \approx 0.32$ s) to meet the end constraints while minimizing the fuel consumption.

In section 7 it is shown how the parallel execution increases the performance of the optimization process.

7 Parallel Execution and Performance Measurements

We have performed measurements for the different algorithms (multiple shooting/collocation and total collocation with variant 1 and 2) applied to the above described applications. The C/C++ source code has been compiled by gcc version 4.6.3 (GCC) with OpenMP support. The measurements are done on an Intel Core i7 CPU 870 with 8 cores @ 2.93 GH (4 real cores and 4 virtual cores).

The corresponding optimization problem is solved by the interior point optimizer Ipopt [16]. **Figure 8** shows the different functions and derivative information that need to be provided to Ipopt for the solution process. In the current implementation the Hessian matrix of the corresponding Lagrangian formulation is calculated numerically by Ipopt. The other information (see **Figure 8**) is provided numerically by external routines. When calculating the Jacobian and Hessian matrices the treatment of the sparsity patterns, is important for the performance of the multiple shooting and total collocation methods [9]. This has been realized for the Jacobian matrix calculation.

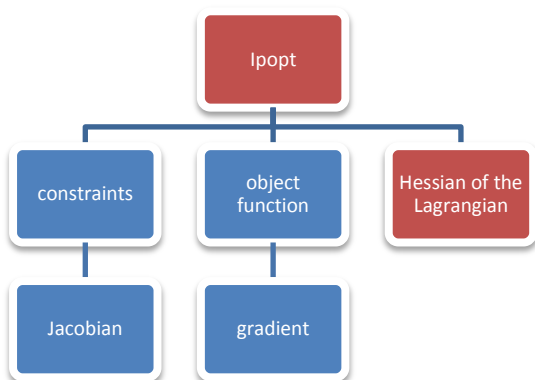


Figure 8. Schematic view of the required components of Ipopt

The multiple shooting algorithm uses an explicit Runge-Kutta formula of order 3 as well as 3 steps within each interval. The multiple collocation method uses 3 intermediate interval points based on Radau formulas. The total collocation uses variant dependent intermediate interval points as described in section 5. The tests

have been performed using 128 intervals when dealing with sparse matrix representation. The user defined functions (see blue boxes of **Figure 8**) have been parallelized.

7.1 Batch Reactor

The speedups obtained and the computation times for the batch reactor are shown in **Table 1** and **Figure 9**.

threads	multiple shooting		multiple collocation	
	lpopt	jac_g	lpopt	jac_g
1	1,5742s	28,93ms	18,47s	343,3ms
2	1,0164s	16,77ms	10,25s	188,3ms
4	0,6691s	9,37ms	5,825s	104,7ms
8	0,6539s	8,52ms	5,055s	89,57ms

Table 1. Computation times for the Jacobian of the constraints and the over-all optimization using multiple shooting/collocation method for the batch reactor

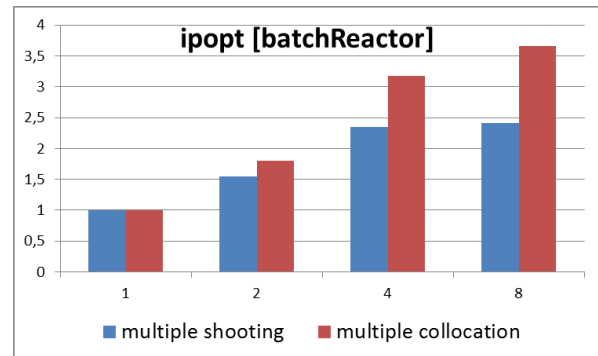


Figure 9. Speedups and computation times of the whole optimization process

Table 1 shows that multiple collocation is much more expensive than the multiple shooting. Reason for this is the computational time needed to solve non-linear systems coming from the implicit discretization. Therefore, by parallelizing the user defined functions a better speedup (**Figure 9**) for the whole optimization can be performed for the multiple shooting method, whereas the speedup for the user defined function (e.g. **Figure 10**) is comparable.

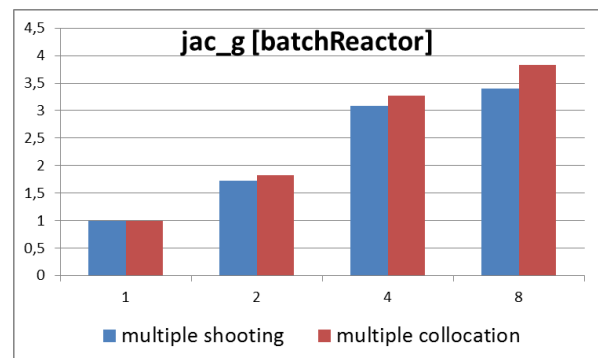


Figure 10. Speedups and computation times for the Jacobian of the constraints

7.2 Diesel Model

The solution process for the diesel model using multiple shooting and multiple collocation is quite time consuming (see **Table 2** and **Table 3**). Especially, the multiple collocation algorithm was only performed with 32 intervals in order to reduce execution time to an acceptable level. Although, parallelization of the user defined function leads to a great speed up, the overall performance of the multiple shooting or collocation method is still poor. The total collocation variants are superior with respect to the over-all performance as can be seen in **Table 3**.

threads	multiple shooting		multiple collocation	
	lpopt	jac_g	lpopt	jac_g
1	1518,4s	1,8196s	368,07s	2,6007s
2	917,17s	0,9671s	196,04s	1,3832s
4	608,29s	0,5286s	108,33s	0,7625s
8	508,71s	0,3861s	87,027s	0,6110s

Table 2. Computation times for the Jacobian of the constraints and the over-all optimization using multiple shooting/collocation method for the diesel model

threads	total collocation 1		total collocation 2	
	lpopt	jac_g	lpopt	jac_g
1	15,40s	8,215ms	14,07s	9,947ms
2	11,49s	4,356ms	10,10s	5,281ms
4	10,19s	2,553ms	8,342s	2,987ms
8	9,452s	1,713ms	7,897s	1,965ms

Table 3. Computation times for the Jacobian of the constraints and the over-all optimization using total collocation method for the diesel model

The speed-up regarding the user-defined function is comparable to the multiple shooting or collocation methods (see **Figure 12**). The speed-up of the whole optimization process is not optimal due to the serial computation and dense treatment of the Hessian matrix calculated internally by Ipopt (see **Figure 11**).

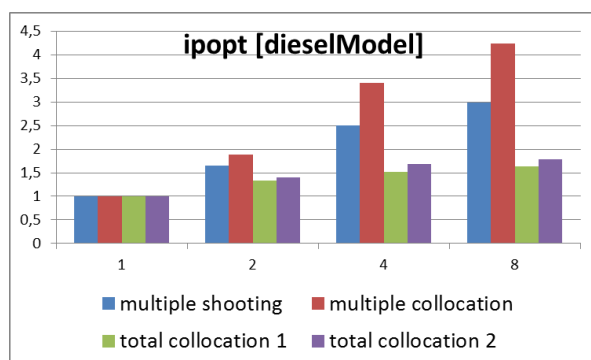


Figure 11. Speedups and computation times of the whole optimization process

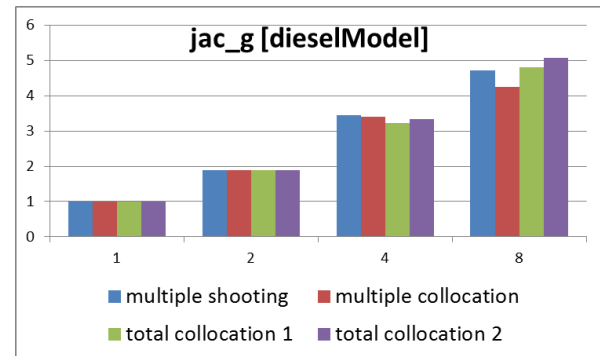


Figure 12. Speedups and computation times for the Jacobian of the constraints

8 Integration with OpenModelica

Support for specifying optimization goal functions and constraints together with Modelica models has now been implemented in OpenModelica. Such integrated models can now be exported via XML to tools such as CasADi [12] which can act as a frontend to ACADO [13].

In the current OpenModelica prototype all aspects of the tool chain are not yet completely implemented. For example, we are currently using numerically derived Gradients, Jacobians and Hessians since the automatic differentiation machinery in OpenModelica has not yet been extended to operate on the optimization problem goal function.

However, the prototype is complete enough to do the measurements of the included model applications on a parallel platform to obtain the speedup curves for parallel execution on 1-8 cores.

The OpenModelica compiler has been extended to export Modelica Models to XML based on an extended version of the FMI XML schema from [14]. The XML export, in addition to the standard Modelica syntax, supports the Optimica extensions from Jmodelica [15]. These extensions allow users to formulate dynamic optimization problems to be solved by a numerical algorithm. The extensions include several constructs including a new specialized class optimization, a constraint section, etc. See the batch reactor example below as well as the Optimica manual for complete information.

```

optimization BatchReactor
  (objective = -x2(finalTime),
   startTime = 0, finalTime = 1)
Real x1(start=1, fixed=true, min=0, max=1);
Real x2(start=0, fixed=true, min=0, max=1);
input Real u(free=true, min=0, max=5);
equation
  der(x1) = -(u+u^2/2)*x1;
  der(x2) = u*x1;
end BatchReactor;

```

The XML generated for flattened Optimica Models can be imported into other non-Modelica Optimization tools like ACADO.

Currently the OpenModelica compiler does not yet use the optimization problem formulation internally as input to automatic differentiation. The Modelica plus Optimica model description is flattened, some common compilation phases are applied e.g. syntax, semantics and type checking, simplification, constant evaluation etc. and then the complete flat model is exported to XML.

9 Conclusions

In this paper parallelized implementations of several different algorithms for solving NOCP have been presented. The well-known multiple shooting or collocation as well as total collocation methods are derived using a general discretization scheme. Total collocation methods have proofed at least in the current implementation and for the tested applications to be superior to the other algorithms.

The corresponding discretized optimization problem has been solved by the interior optimizer Ipopt. Further speedup of the optimization process for all described algorithms have been achieved by parallelizing the calculation of model specific parts (e.g. constraints, Jacobians, etc.). So far the evaluation of derivatives have been done numerically. This will be further improved using the already available symbolic differentiation capabilities of OpenModelica [11]. Finally, this work will be continued by applying the proposed algorithms on more industrial relevant applications together with a thorough testing on advanced parallel hardware architectures.

10 Acknowledgements

This work has been partially supported by Serc, by SSF in the EDOp project and by Vinnova as well as the German Ministry BMBF (BMBF Förderkennzeichen: 01IS09029C) in the ITEA2 OPENPROD project. The Open Source Modelica Consortium supports the OpenModelica work.

References

- [1] Open Source Modelica Consortium. *OpenModelica System Documentation Version 1.8.1*, April 2012. <http://www.openmodelica.org>
- [2] Modelica Association. *The Modelica Language Specification Version 3.2*, March 24th 2010. <http://www.modelica.org>. Modelica Association.
- [3] Jasem Tamimi, Pu Li. A combined approach to nonlinear model predictive control of fast systems. *Journal of Process Control*, 20, pp 1092–1102, 2010.
- [4] Biegler, Lorenz T. 2010. *Nonlinear Programming: Concepts, Algorithms, and Applications to Chemical Processes*. s.l. : Society for Industrial Mathematics, 2010.
- [5] Munz, Claus-Dieter and Westermann, Thomas. 2009. *Numerische Behandlung gewöhnlicher und partieller Differenzialgleichungen*. Berlin Heidelberg : Springer Verlag, 2009
- [6] Heuser, Harro. 2006. *Gewöhnliche Differentialgleichungen*. Wiesbaden : Teubner Verlag, 2006.
- [7] Tamimi, Jasem. 2011. *Development of Efficient Algorithms for Model Predictive Control of Fast Systems*. Düsseldorf: VDI Verlag, 2011.
- [8] Friesz, Terry L. 2007. *Dynamic Optimization and Differential Games*. US: Springer US, 2007.
- [9] Folkmar, Bornemann und Deufhard, Peter. 2008. *Numerische Mathematik: Numerische Mathematik 2: Gewöhnliche Differentialgleichungen: Bd II: [Band] 2*. s.l. : Gruyter, 2008.
- [10] Martin Sivertsson and Lars Eriksson Optimal power response of a diesel-electric powertrain. Submitted to ECOSM'12, Paris, France, 2012.
- [11] Braun, Willi, Ochel Lennart and Bachmann Bernhard. *Symbolically Derived Jacobians Using Automatic Differentiation - Enhancement of the OpenModelica Compiler*, Modelica Conference 2011
- [12] Joel Andersson; Johan Åkesson; Moritz Diehl, CasADi - A symbolic package for automatic differentiation and optimal control, Proc. 6th International Conference on Automatic Differentiation, 2012.
- [13] Houska, B., Ferreau, H.J., and Diehl, M. (2011). ACADO toolkit - an open source framework for automatic control and dynamic optimization. *Optimal Control Applications & Methods*, 32(3), 298-312.
- [14] Functional Mock-up Interface: <http://www.functional-mockup-interface.org/index.html>
- [15] Johan Åkesson. Optimica—An Extension of Modelica Supporting Dynamic Optimization. In 6th International Modelica Conference 2008. Modelica. Association, March 2008
- [16] Interior Point OPTimizer (Ipopt) <https://projects.coin-or.org/Ipopt>

11 Appendix A

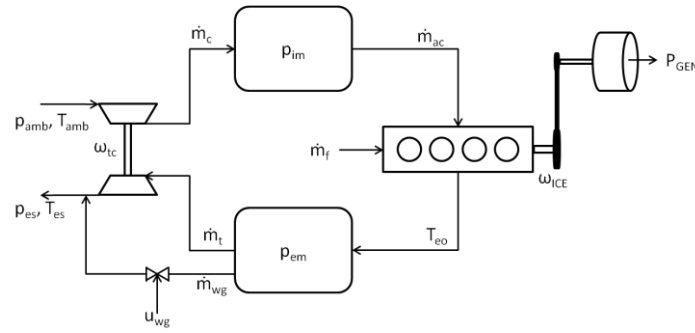


Figure 13. Diesel Engine Model

Powertrain model

$$\dot{\omega}_{ice} = \frac{P_{ice} - P_{gen}}{\omega_{ice} J_{genset}}$$

Intake System

- Compressor

$$\Pi_c = \frac{p_{im}}{p_{amb}}, \quad \Pi_{c,max} = \left(\frac{\omega_{ice}^2 R_c^2 \psi_{max}}{2c_p T_{amb}} + 1 \right)^{\frac{\gamma_a}{\gamma_a - 1}}, \quad \dot{m}_{c,corr} = \dot{m}_{c,corr,max} \sqrt{1 - \left(\frac{\Pi_c}{\Pi_{c,max}} \right)^2},$$

$$\dot{m}_c = \frac{\dot{m}_{c,corr} p_{amb}/p_{ref}}{\sqrt{T_{amb}/T_{ref}}}, \quad P_c = \frac{\dot{m}_c c_{pa} T_{amb} (\Pi_c^{\frac{\gamma_a}{\gamma_a - 1}} - 1)}{\eta_c}, \quad \eta_c = c_c$$

- Intake manifold

$$\dot{p}_{im} = \frac{R_a T_{im}}{V_{is}} (\dot{m}_c - \dot{m}_{ci}), \quad T_{im} = T_{cool}$$

Cylinder

- Gas Flow

$$\dot{m}_{ci} = \frac{\eta_{vol} p_{im} \omega_{ice} V_D}{4\pi R_a T_{im}}, \quad \eta_{vol} = c_{vol}, \quad \dot{m}_f = \frac{10^{-6}}{4\pi} u_f \omega_{ice} \eta_{cyl}, \quad \lambda = \frac{\dot{m}_{ci}}{\dot{m}_f (A/F)_s}$$

- Torque

$$T_{ice} = T_{ig} - T_{fric} - T_{pump}, \quad T_{pump} = \frac{V_D}{4\pi} (p_{em} - p_{im}), \quad T_{ig} = \frac{u_f 10^{-6} n_{cyl} q_{HV} \eta_{ig}}{4\pi}, \quad \eta_{ig} = \eta_{ig,ch} \left(1 - \frac{1}{r_c^{\gamma_{cyl} - 1}} \right)$$

$$T_{fric} = \frac{V_D}{4\pi} 10^5 \left(C_{fr1} \left(\frac{\omega_{ice} \frac{60}{2\pi}}{1000} \right)^2 + C_{fr2} \left(\frac{\omega_{ice} \frac{60}{2\pi}}{1000} \right)^2 + C_{fr3} \right)$$

- Temperature

$$\Pi_c = \frac{p_{em}}{p_{im}}, \quad q_{in} = \frac{\dot{m}_f q_{HV}}{\dot{m}_f + \dot{m}_{ac}}, \quad x_p = \frac{p_3}{p_2} = 1 + \frac{q_{in} x_{cv}}{c_{va} T_{im} r_c^{\gamma_a - 1}}$$

$$T_{eo} = \eta_{sc} \Pi_e^{(1 - \frac{1}{\gamma_a})} r_c^{(1 - \gamma_a)} x_p^{\frac{1}{\gamma_a - 1}} \left(q_{in} \left(\frac{1 - x_{cv}}{c_{pa}} + \frac{x_{cv}}{c_{va}} \right) + T_{im} r_c^{(\gamma_a - 1)} \right)$$

Exhaust System

- Exhaust Manifold:

$$\dot{p}_{em} = \frac{R_e T_{em}}{V_{em}} (\dot{m}_{ci} + \dot{m}_f - \dot{m}_t - \dot{m}_{wg}), \quad T_{em} = T_{eo}$$

- Turbine

$$\Pi_t = \frac{p_{es}}{p_{em}}, \quad \Pi_t^* = \max \left(\sqrt{\Pi_t}, \left(\frac{2}{\gamma_e - 1} \right)^{\frac{\gamma_e}{\gamma_e - 1}} \right), \quad \psi_t(\Pi_t^*) = \sqrt{\frac{2\gamma_e}{\gamma_e - 1} \left((\Pi_t^*)^{\frac{2}{\gamma_e}} - (\Pi_t^*)^{\frac{\gamma_e + 1}{\gamma_e}} \right)},$$

$$\dot{m}_t = \frac{p_{em}}{\sqrt{R_e T_{em}}} \psi_t A_{t,eff}, \quad P_t = \dot{m}_t c_{pe} T_{em} \eta_t \left(1 - \Pi_t^{\frac{\gamma_e - 1}{\gamma_e}} \right), \quad \eta_t = c_t, \quad J_{tc} \dot{\omega}_{tc} = \frac{P_t - P_c}{\omega_{tc}} - \omega_{fric} \omega_{tc}^2$$

- Wastegate

$$\Pi_{wg} = \frac{p_{es}}{p_{em}}, \quad \Pi_{wg}^* = \max \left(\Pi_{wg}, \left(\frac{2}{\gamma_e + 1} \right)^{\frac{\gamma_e}{\gamma_e + 1}} \right), \quad \psi_{wg} = \sqrt{\frac{2\gamma_e}{\gamma_e - 1} \left((\Pi_{wg}^*)^{\frac{2}{\gamma_e}} - (\Pi_{wg}^*)^{\frac{\gamma_e + 1}{\gamma_e}} \right)}, \quad \dot{m}_{wg} = \frac{p_{em}}{\sqrt{R_e T_{em}}} \psi_{wg} u_{wg} A_{wg,eff}$$

Model Constants

Symbol	Description	Value	Unit
p_{amb}	Ambient pressure	1.011e5	Pa
T_{amb}	Ambient temperature	298.46	K
c_{pa}	Specific heat capacity of air, constant pressure	1011	J/(kg.K)
c_{va}	Specific heat capacity of air, constant volume	724	J/(kg.K)
γ_a	Specific heat capacity ratio of air	1.3964	-
R_a	Gas constant, air	287	J/(kg.K)
c_{pe}	Specific heat capacity of exhaust gas, constant pressure	1332	J/(kg.K)
γ_e	Specific heat capacity ratio of exhaust gas	1.2734	-
R_e	Gas constant, exhaust gas	286	J/(kg.K)
γ_{cyl}	Specific heat capacity ratio of cylinder gas	1.35004	-
T_{im}	Intake manifold temperature	300,6186	K
p_{es}	Pressure in exhaust system	1.011e5	Pa
$(A/F)_s$	Stoichiometric oxygen-fuel ratio	14.54	-
q_{HV}	Diesel heating value	42.9e6	J/kg

Model Parameters

Symbol	Description	Value	Unit
n_{cyl}	Number of cylinders	6	-
V_D	Engine displacement	0.0127	m^3
r_c	Compression ratio	17.3	-
J_{genset}	Inertia of the engine-generator	3.5	kgm^2
V_{is}	Volume of intake system	0.0218	m^3
R_c	Compressor radius	0.04	M
ψ_{max}	Max. compressor head parameter	1.5927	-
$\dot{m}_{c,corr,max}$	Max. corrected compressor mass flow	1.2734	-
η_c	Compressor efficiency	286	J/(kg.K)
η_{vol}	Volumetric efficiency	1.35004	-
$\eta_{ig,ch}$	Combustion chamber efficiency	0.6774	-
c_{fr1}	Friction efficiency	1.011e5	Pa
c_{fr2}	Friction efficiency	14.54	-
c_{fr3}	Friction efficiency	42.9e6	J/kg
η_{sc}	Non-ideal Seliger cycle compensation	1.054	-
x_{cv}	Ratio of fuel burnt during constant volume	0.4046	-
V_{em}	Volume of exhaust manifold	0.0199	m^3
J_{tc}	Turbocharger inertia	1.9662 e-4	kgm^2
ω_{fric}	Turbocharger friction	2.4358 e-5	kgm^2/rad
$A_{t,eff}$	Effective turbine area	9.8938 e-4	m^3
η_t	Turbine efficiency	0.7278	-
$c_{wg,1}$	Wastegate parameter	0.6679	-
$c_{wg,2}$	Wastegate parameter	5.3039	-
$A_{wg,eff}$	Effective wastegate area	8.8357 e-4	m^3

Optimization Library for Interactive Multi-Criteria Optimization Tasks

A. Pfeiffer

Institute of System Dynamics and Control, German Aerospace Center DLR, Oberpfaffenhofen
Andreas.Pfeiffer@dlr.de

Abstract

The commercial library Optimization 2.1 for interactive multi-criteria optimization tasks has been released along with Dymola 2013. The library offers several numerical optimization algorithms for solving different kinds of optimization tasks. User defined Modelica functions or models provide the basis for an interactive optimization process where the user keeps overview of complex multi-criteria optimization tasks that can take discrete parameters, several model operating points or trajectories into account. Computational performance of optimization runs can be significantly increased by parallel numerical integrations of the Modelica model on multi-core machines.

Keywords: Modelica; Optimization; Multi-Criteria; Trajectory Optimization; Parallel Simulation

1 Introduction

In principle, numerical optimization algorithms may be very powerful tools in engineering design processes like modeling, model validation or controller design. However, the fact that numerical algorithms are available does not necessarily encourage engineers to apply them. A user-friendly, easy handling of a well integrated optimization tool is necessary to make the advantages of automatic optimization available for non-experts. The presented Optimization library realizes this requirement in the Modelica world when working with Dymola [DS12b] or CATIA [DS12a].

1.1 Related Work

OMOptim [TNT+11] is an initiative to provide an open source optimization platform within OpenModelica. The emphasis of this platform is on using genetic algorithms, whereas interfacing gradient based optimization methods is planned for the future. The application is currently tailored to optimize model parameters of Modelica models. The library presented in the paper at hand provides a variety of different

optimization tasks solved by several sophisticated local and global optimization algorithms.

In JModelica.org the Modelica extension Optimica is supported to solve dynamic optimization [AAG+10]. The approach in Optimica is different to the presented one, because Optimica defines additional Modelica language elements to describe Optimization problems directly in Modelica. Consequently, special compilers are needed to generate code for the optimization runs. JModelica.org supports collocation methods for dynamic optimizations. In the presented approach, (standard) Modelica models are compiled by Dymola. The well-proven numerical integration algorithms provided by Dymola are used in the optimization loop. Tailored graphical user interfaces support the user in several optimization tasks.

The library `Design.Optimization` [EOM+05] is the forerunner of the presented library. For the new version the library has been completely reimplemented with many new features. The new concept of different optimization *tasks* is enhanced by specialized graphical user interfaces (GUIs). The primary concept and the code of numerical algorithms for solving multi-criteria optimization problems are based on [JBL+02].

1.2 Optimization Problem Formulation

The multi-criteria optimization problems considered in the Optimization library can be formulated as follows:

$$\min_{p \in B} f(\text{diag}(d_1)^{-1} c_1(p))$$

$$\text{such that } c_2(p) \leq d_2, \quad c_3(p) = d_3$$

$$\text{with } c = \begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix}, \quad d = \begin{pmatrix} d_1 \\ d_2 \\ d_3 \end{pmatrix} \text{ and}$$

$$f = \begin{cases} \max & \dots \text{ maximum of criteria values, or} \\ \|\cdot\|_2^2 & \dots \text{ sum of squared criteria values, or} \\ \|\cdot\|_1 & \dots \text{ sum of absolute criteria values.} \end{cases}$$

Free parameters p (e.g. some Modelica parameters in models) to be varied during the optimization process are called *tuner parameters* or tuners. The first part of the *criteria* vector c represents the objectives of the optimization (e.g. the overshoot of a variable in a model). The goal is to minimize all these objectives. The criteria components that define inequality or equality constraints are optional. They enable formulation of conditions on some criteria components if needed. The demand values d serve as reciprocal scaling factors of the criteria. They enable a different weighting of the individual criteria to be minimized. The tuner box B defines minimum and maximum values for each tuner parameter, thus limiting the range in which the tuner parameters can be varied.

For multi-criteria optimization problems a whole set of optimal solutions generally exists: the *Pareto* optimal solutions [E05]. For these solutions it is not possible to decrease one of the components of the objectives vector c_1 without increasing another one. It means the different criteria conflict each other. Finding all Pareto optimal solutions requires very high computational effort. In many cases it is sufficient to transform a multi-criteria problem to an optimization problem with a scalar objective function f . This approach is applied to the Optimization library with the maximum of the objectives, the sum of the squares of the objectives or the sum of the absolute values of the objectives.

1.3 Discrete Tuner Parameters

Discrete tuners are tuners that only have a finite number of values to be set. Examples for such tuners are configuration parameters that represent different topologies, e.g. switching modes in networks.

Three possibilities are available to define discrete tuners in the Optimization library. At the level of each tuner parameter, one can define the number of equidistant discrete values within the interval $[\min, \max]$. Only these points can be selected by the optimization algorithm to set the tuner value.

name	min	max	equidistant	discreteValues
"Kf"	-10	0	0	{-7.8, -2.5, -9.3}
"Ki"	-10	0	6	fill(0, 0)
"Kq"	0	10	0	fill(0, 0)

Figure 1: Discrete values for tuner parameters in the optimization setup GUI.

For example, setting *equidistant* = 6 for *min* = -10, *max* = 0 enables the values -10, -8, -6, -4, -2, 0 for

the tuner K_i in Figure 1. The second possibility to define discrete tuners is to give a Modelica vector of values that can be set to the tuner parameter, e.g. *discreteValues* = {-7.8, -2.5, -9.3} for tuner parameter K_f .

At the level of all tuner parameters a list of values of discrete tuner parameter sets can be defined in a matrix. Each column corresponds to a tuner parameter, see Figure 2. It is possible to simply import the matrix from and export it to file. This feature allows to automatically evaluate a long list of tuner values generated by a separate tool.

Rows 2		Kf	Ki	Kq
1		-7.8	-2.54	6.1
2		-3.4	-8	2.883

Figure 2: Discrete tuner matrix in the optimization setup GUI.

1.4 Optimization and Evaluation Algorithms

The following numerical optimization algorithms are available in the Optimization library: *Sequential Quadratic Programming (SQP)*, *Quasi Newton (BFGS) method*, *Pattern Search*, *Simplex Method* and *Genetic Algorithm*. SQP and BFGS algorithms rely on derivatives of the criteria with respect to the tuner parameters and have good convergence properties for smooth optimization problems. Pattern Search and Simplex Method are more robust against nonsmoothness but generally need more criteria evaluations to converge. Genetic Algorithm is the only approach to find a global solution whereas the others are local convergent methods. Further details to the implemented optimization algorithms can be found in [J11].

All the optimization algorithms have in common that they work more or less sequentially. Most values for tuners depend on criteria values of previous evaluations. So, there are limited possibilities to parallelize the (time consuming) evaluations of criteria. In contrast to these algorithms, pure evaluation methods independently set tuner values at the beginning of the process. Of course, constraints fulfillment is therefore not guaranteed.

Two evaluation methods are implemented in the Optimization library: *Random Search* and *Systematic Tuner Variation*. Random Search takes uniformly distributed random values between minimum and

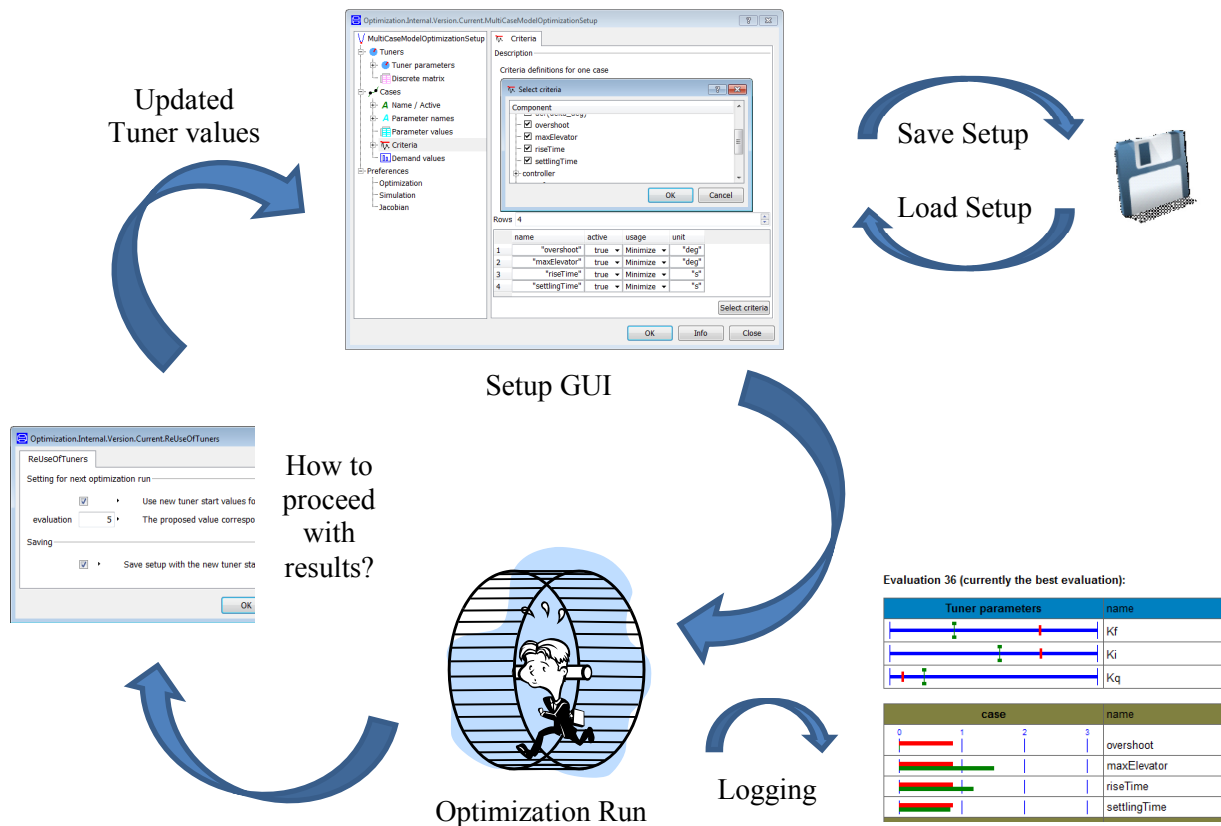


Figure 3: Optimization process for GUI supported Optimization tasks.

maximum of each tuner parameter. Systematic Tuner Variation is based on discrete tuners. If the discrete tuner matrix is activated, the corresponding tuner values are used row by row of the matrix. If the discrete tuner matrix is not used, all combinations of equidistant or given discrete tuner values are the basis for the criteria evaluations. For the example in Figure 1 there are $3 \cdot 6 \cdot 1 = 18$ different sets of discrete tuner values.

Table 1: Overview of the optimization and evaluation algorithms with their capability to support continuous and / or discrete tuners.

Algorithm	Continuous	Discrete	Mixed
SQP	✓		
BFGS	✓		
Pattern Search	✓		
Simplex Meth.	✓		
Genetic Alg.	✓	✓	✓
Random Search	✓	✓	✓
Systematic Var.		✓	

Most of the interfaced algorithms are designed to handle continuous tuner parameters. It means that the tuner values can be arbitrarily varied inside a given

interval. Table 1 gives an overview which algorithm also supports discrete tuners or problems with both continuous and discrete tuner parameters.

1.5 Optimization Process

For each of the GUI supported optimization tasks the process to configure the task, to start the optimization and to handle the results is nearly the same and is discussed in the following by means of Figure 3.

By starting the corresponding setup GUI for an optimization task, the user gets a hierarchical list of settings to be configured. For each task one has to specify tuners and criteria depending on the type of the task. For optimization tasks requiring a model, additional settings for the model simulation have to be provided. All the information given in the setup GUI can be saved to a Modelica file. The file contains a call starting the corresponding setup GUI filled with the saved entries. Of course, the textual file can be edited before starting the setup GUI. So, loading an optimization setup is simply running the Modelica function generated when saving the setup.

After the optimization setup is configured, the optimization run is started. During the run the current

solutions may be logged to an HTML-file, also interactively displayed in Dymola's Command window. The logging has two intentions. Firstly, the history of a complete optimization run can be reconstructed. Secondly, optimization runs may last hours or days. It is very important in these cases to have a feedback, what the optimization algorithm is currently doing, to quickly react on non-intended intermediate optimization results. The HTML-logging lists the current tuner and criteria values and visualizes them in different colors in comparison to values at the beginning of the optimization.

Beside the HTML-logging there is a logging of pure numeric data to be processed after the optimization run if it is necessary. After the optimization run is finished, the user is asked how he wants to proceed. There is the possibility to reset the tuner parameters by values generated by the optimization process. For example, one can select the tuner values of the best evaluation (= solution) of the optimization run. These settings can be used to proceed the optimization process with different settings, e.g. using another optimization algorithm. In any case, after an optimization run the setup GUI is displayed (with possibly changed tuner values) and can be configured as described above.

2 Function based Optimization

Two optimization tasks based on user-defined Modelica functions are described. Whereas *Function Optimization* is an interactive task, *Realtime Optimization* is designed to be called in model equations during the numerical integration.

2.1 Function Optimization

The task *Function Optimization* is designed for the most general case of an optimization problem in Modelica. The user has to provide a Modelica function that evaluates the criteria (and constraints) functions. Optionally, a user-defined function for the evaluation of the Jacobian matrix can be incorporated. The task can be used for simple academic optimization problems resulting in a criteria function of a few lines of code, or for every complex optimization problem including simulations and linearizations of several models. The user has to program and control the simulations and linearizations by available functions in Modelica and Dymola.

The main part of a function optimization problem is to program the criteria function in Modelica. The

criteria function returns a criteria vector depending on the tuner values. The criteria can either be parts of the optimization's objective function or be one of the constraints of the optimization problem. A criteria function has to have defined interface variables from the partial function `PartialCriteriaVariables`:

```
partial function PartialCriteriaVariables
  input Real tuners[:];
  output Real criteria[:];
end PartialCriteriaVariables;
```

A typical criteria function looks like the following prototype. One can add own input variables to the criteria function. The values for these inputs have to be declared in the name of the criteria function in the setup, e.g. "myCriteriaFunc(myVar=<value>)".

```
function myCriteriaFunction
  extends PartialCriteriaVariables;
  input <AnyType> myVar;
algorithm
  criteria := ... (tuners, myVar);
end myCriteriaFunction;
```

Gradient based optimization algorithms (SQP, BFGS) need the Jacobian matrix of the criteria with respect to tuner parameters. The user can select between symmetric finite differences and forward difference quotients. There is also the possibility to program the Jacobian matrix by oneself, e.g. if one knows the analytical Jacobian matrix. The interface variables are defined in the following partial function:

```
partial function PartialJacobianVariables
  input Real tuners[:];
  input PartialCriteriaVariables CritFunc;
  output Real Jacobian[:,size(tuners,1)];
end PartialJacobianVariables;
```

To a Jacobian function one can also add own input variables, see the following prototype of a typical Jacobian function:

```
function myJacobianFunction
  extends PartialJacobianVariables;
  input <AnyType> myVar;
algorithm
  Jacobian := ... (tuners, myVar);
end myJacobianFunction;
```

2.2 Realtime Optimization

Realtime Optimization is in some way different to the other optimization tasks. Realtime Optimization provides the framework for an optimization function to be called during the numerical integration of a

model. A possible application of this optimization task is a discrete controller that solves an optimization problem to predict new controller values every sample time. The optimization problem itself is very similar to that of *Function Optimization*. User defined functions for criteria evaluation and optional functions for the Jacobian matrix provide the basis for the optimization task. Because Realtime Optimization is active during a simulation many times, there is no GUI support for it. A Modelica model calling the optimization function typically has the following structure:

```

model myModel
  Real resultTuners[...];
  Real resultCriteria[...];
  KernelProblem problem(...);
  ...
equation
  ...
  when sample(0, 0.1) then
    (resultTuners,resultCriteria) =
      run(problem, CriteriaFunc =
          function myCriteriaFunction);
  end when;
  ...
end myModel;

```

At each sample point the optimization run is started by the function *run*. The optimization problem is described by the record *problem* that includes the tuner and criteria definitions as well as the optimization options. The approach is currently used in model predictive control for an electric vehicle [K10].

3 Model based Optimization

This section deals with optimization tasks based on the numerical integration of a Modelica model. The computation of the optimization criteria is part of the numerical integration. Because model simulation is the main application of dealing with Modelica models, the following optimization tasks and their features may be considered as the core of the Optimization Library.

3.1 Criteria Library

To support all model based optimization tasks the sub-library `Optimization.Criteria` is part of the whole package. The library (see Figure 4) provides models that compute typical criteria from time dependent model variables. The collection of criteria models helps the user to prepare his system model for conducting an optimization on it. For Real signals the following models are included: minimum,

maximum, mean value, moving average and integral norm. In Figure 5, some examples are illustrated. Computing deviations between two signals may be handled by the corresponding criteria models. In the field of controller design typical design criteria are *overshoot*, *rise time* and *settling time*. Each of them is represented by a corresponding criteria model. Some of the criteria models require the input signals to be differentiated.

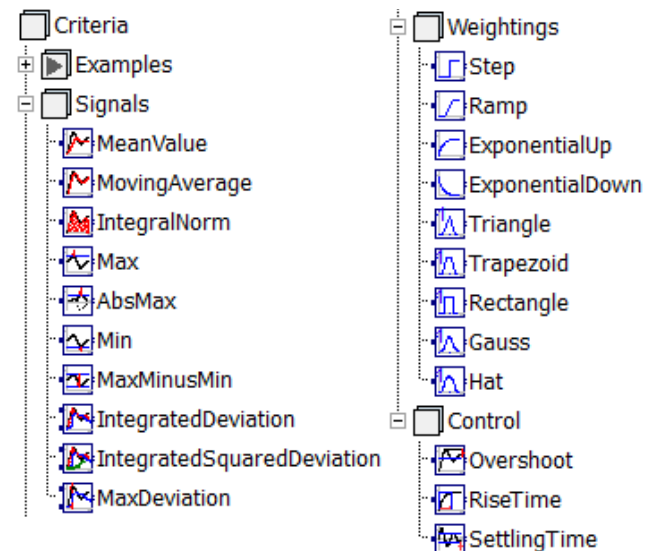


Figure 4: Criteria library.

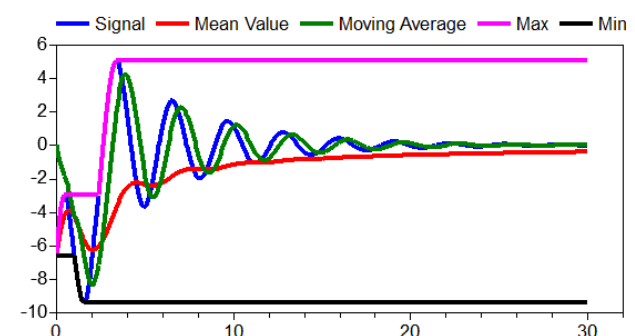


Figure 5: Typical signals of criteria models.

In some cases only parts of the whole time interval shall be used to compute a certain criterion, or some time areas shall be weighted more than others. For such needs several weighting models are provided: *Step*, *Ramp*, *Triangle*, etc.

3.2 Model Optimization

The task *Model Optimization* is designed to optimize parameters of a Modelica model. The user can select from a list of model parameters to define tuners, see Figure 6. Also it is possible to get a list of all time depending model variables to be selected for criteria

variables. The value of the criterion is defined by the final value of the criterion variable at the end of the integration interval.

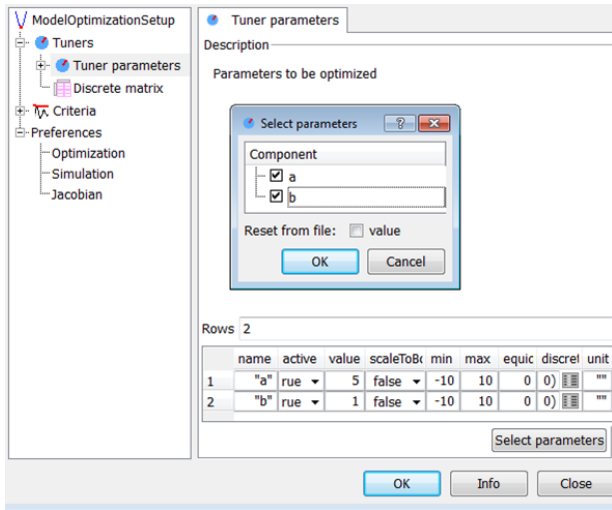


Figure 6: GUI for selecting model parameters as tuners.

The simulation of the model to be optimized has to be specified by usual simulation preferences like start and stop time or the numerical integration algorithm. Additionally, different modes to accelerate the numerical integration of the model equations are implemented, see Section 3.5.

A typical application of Model Optimization is the identification of model parameters by comparing simulation results and corresponding measurements from a test bench. A further application is well known in the field of controller synthesis. To improve the controller performance automatic optimization is applied to the system model.

3.3 Multi Case Model Optimization

Multi Case Model Optimization is an extension of the task *Model Optimization* and has its origin in the field of model based controller design. Most controllers do not only have to guarantee performance and stability of a system in one, but in several operating points. The optimization of the controller parameters includes the simulation of a system model in different operating points that are characterized by different values of special model parameters, the *case parameters*. These model parameters are disjoint with the tuner parameters and are not varied by the optimization algorithm. The different model simulations that are defined by the case parameters are called *cases*. In Figure 7 main parts of the corresponding task setup GUI are shown.

Each case should have a name to distinguish it from the other cases. In Figure 7 there are three cases: *nominal*, *worstOvershoot* and *worstSettlingTime*. The case parameters (e.g. *Ma*, *Md*, ..., *Zd*) can be selected from a list of all independent model parameters. For each case every case parameter gets a value, see the matrix in Figure 7. The model is simulated with these case parameter values for each case. The criteria of the optimization task are similarly specified as for the task Model Optimization.

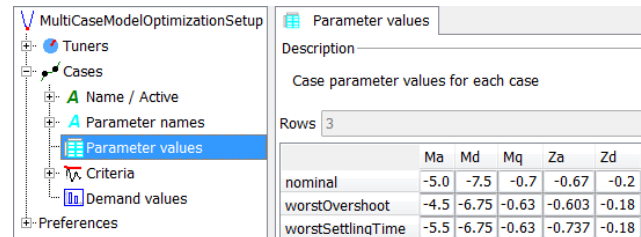


Figure 7: Optimization setup GUI for Multi Case Model Optimization.

In summary, every case contributes to the overall criteria vector of the optimization problem, see Figure 8 for an example. Depending on the objective function type all these criteria values are combined to the objective function value. In the example the value is the maximum of all criteria values: *riseTime* for the case *worstSettlingTime*.

nominal				name	scf
0	1	2	3	overshoot	0
[Bar chart]				maxElevator	0.91217
[Bar chart]				riseTime	0.89220
[Bar chart]				settlingTime	0.78246
worstOvershoot				name	scf
0	1	2	3	overshoot	0.94221
[Bar chart]				maxElevator	0.94158
[Bar chart]				riseTime	0.86281
[Bar chart]				settlingTime	0.74443
worstSettlingTime				name	scf
0	1	2	3	overshoot	0
[Bar chart]				maxElevator	0.94251
[Bar chart]				riseTime	0.94253
[Bar chart]				settlingTime	0.94251

Figure 8: Logging of multi case criteria.

3.4 Trajectory Optimization

Problems of Optimal Control arise in different fields of applications. The goal is to minimize an objective functional with respect to one or more time dependent control trajectories. Various constraints are typical for optimal control problems. Dynamic model equations appear in most of the problems in technical

applications. Consequently, an optimization task *Trajectory Optimization* is provided in the library.

There are many techniques [B01] to numerically solve an infinite dimensional optimal control problem. In the Optimization library the solution procedure is according to the task *Model Optimization*. It means that tuners are varied by the optimization algorithm and for each computation of the criteria a model simulation is performed. This *Single Shooting Technique* is based on a finite dimensional optimization problem approximating the original problem.

The control trajectories are approximated by *B-splines* of degree k . The number of samples N and the interpolation degree k define the construction of a B-spline as control trajectory [DH02]. The B-spline has N equidistant knots on the time interval the spline is defined (normally this is the integration interval of the model). Further there are $N + k - 1$ *de Boor control points* that parameterize the spline. A spline $s(t)$ is a piecewise polynomial function between the knots. The individual polynomials have at most the degree k . The polynomials are appended such that the complete spline is $k - 1$ times continuously differentiable on the whole interval it is defined. Because a B-spline is contained in the convex hull of its de Boor points m , the box constraints $u_{\text{Min}} \leq m \leq u_{\text{Max}}$ (for lower and upper bounds u_{Min} , u_{Max}) are valid for the whole spline function: $u_{\text{Min}} \leq s(t) \leq u_{\text{Max}}$. Therefore, the control points m are selected as tuners to be varied by the optimization algorithm.

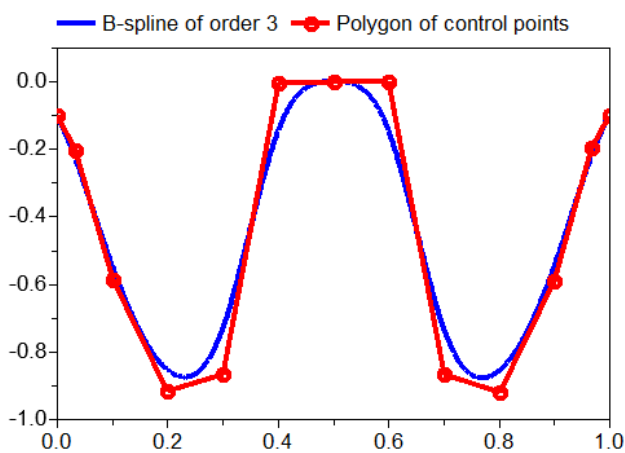


Figure 9: Polygon of B-spline control points and corresponding B-spline trajectory.

In Figure 9 the polygon of 13 control points and the corresponding B-spline of degree 3 ($N = 11$, $k = 3$) are shown. The control points correspond to the 13 time values 0.0, 0.033, 0.1, 0.2, ..., 0.8, 0.9, 0.967,

1.0. Additional to the given time grid 0.0, 0.1, ..., 1.0 there are two values at the boundaries: 0.033 and 0.967. They represent the free boundary conditions of the B-spline.

The optimization setup for Trajectory Optimization includes the selection of model inputs that represent the control trajectories. For these trajectories the number of sample points N and the interpolation degree k has to be specified by the user. Any starting trajectory may be provided in a separate file. An example using the Trajectory Optimization task is given in Section 4.

3.5 Parallel Numerical Integration

Because the numerical integration of model equations normally is the most time intensive part of any model based optimization tasks, several techniques are applied to reduce the computation time of the numerical integration inside the optimization loop. The default case is a sequential execution of the numerical integration runs by calling Dymola's simulation executable for each new set of model parameters. We call it *single* simulation technique.

An optimized version of sequential integration runs is provided by Dymola. The executable is started only one time and independent model parameter values are sequentially read from file and processed by the numerical integration. Especially for many simulation runs with very short elapsed real times for one model simulation, this *multi* simulation approach accelerates the numerical integration in summary, because process overhead is avoided.

Independent simulation runs of a model may be executed in *parallel*. Especially for multi-core machines this may reduce the computation time of the whole optimization run. In the Optimization library the simulation runs are parallelized in different threads by calling several copied simulation executables in an OpenMP program. OpenMP is a software interface for shared-memory parallel programming on different platforms. It is supported by many computer hardware and software vendors [CJP08]. For parallel simulations the user can specify the number of threads up to the double of the number of available cores. Table 2 shows execution sequences for different simulation modes in principle.

To measure the acceleration in computation time, a test is performed for different simulation modes. The model `Electrical.Analog.Examples.Rectifier`

from the Modelica Standard Library 3.2 is simulated 1000 times with identical parameter values. To increase the elapsed real time of one numerical integration run, the stop time of the integration is increased. The test is executed on a PC with an Intel Xeon X5550 quad-core processor (2.67 GHz) with activated hyper-threading.

Table 2: Execution sequence for single, multi and parallel (with 3 threads) simulations.

Execution sequence ↓	Single	Multi	Parallel 3		
			Thread 1	Thread 2	Thread 3
1	Sim. 1				
2	Sim. 2	Sim. 1			
3	Sim. 3	Sim. 2	Sim. 1	Sim. 3	Sim. 5
4	Sim. 4	Sim. 3	Sim. 2	Sim. 4	
5	Sim. 5	Sim. 4			
6		Sim. 5			

In Figure 10 the results of the test are illustrated. Depending on the execution time for one model simulation, the speed factor with respect to the single simulation technique is plotted for multi and parallel simulations. Parallel simulations are performed with 2, 4 and 8 threads. For very fast model simulations the multi simulation approach is clearly superior. Compared to single simulation the multi simulation is up to 4 times faster although no parallelization technique is applied. The parallel execution of 1000 model simulations results in maximum speed factors of 1.9, 3 and 4 for 2, 4 and 8 threads. These maximum factors are reached if the execution time for one model simulation is greater than 1 second. Below this bound the speed factor is decreasing due to the process overhead. For machines with many cores the limiting influence for parallelization is probably memory access.

An important assumption for the performance test is the independency of all evaluated model parameter values. The Optimization library supports two algorithms that fulfill this assumption: Random Search and Systematic Tuner Variation (see Section 1.4). For these algorithms the tuner values of all evaluations may be determined before running any simulation, therefore full parallel evaluations are possible. So, speed factors as shown in Figure 10 can be reached.

Accelerating the computation time in nonlinear optimization by parallel evaluations of the criteria has

been investigated since several years, e.g. see [LAS97]. The optimization algorithms of the Optimization library partially support parallel criteria evaluations. During an optimization run there are both evaluations of the criteria that can be parallelized and such ones that cannot be parallelized. The evaluation of numerical Jacobian matrices typically needs the most computation time for optimization runs with SQP and BFGS methods. Consequently, the Optimization library supports computing numerical Jacobian matrices by parallel model simulations. It also supports parallel criteria evaluations of multi case optimization tasks (see Section 3.3). The simulation runs of a model with different case parameter values are independent and therefore can be computed in parallel. It is planned to support parallel model simulations for independent criteria evaluations of the genetic algorithm.

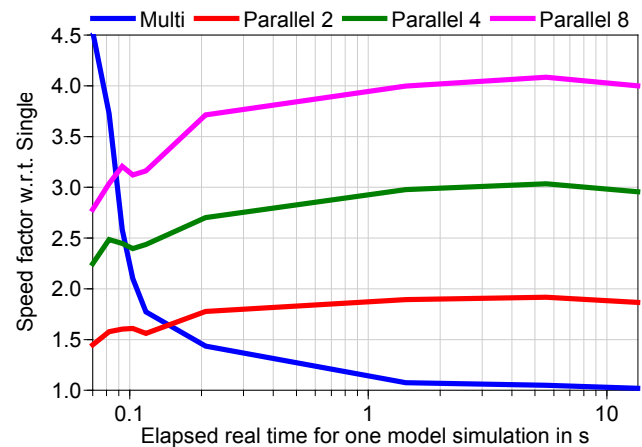


Figure 10: Speed factors for different simulation modes.

Depending on the used optimization algorithm, the Modelica model and the number of tuners, the speed factor for a *complete optimization run* differs. On the test machine a factor of 3 in computation time has been observed for optimization examples using a model that needs more than 1 second of elapsed real time per simulation, see Section 4.3 for an example.

4 Application Example

In [EOM+05] the full robot model of the Modelica standard library is used to demonstrate a multi case optimization for controller design. Of course, the current version of the Optimization library can still handle this kind of optimization task (see Section 3.3). In the following a trajectory optimization setup for the robot model is presented to find reference trajectories for the robot's movements from one point to another point in space.

4.1 Robot Model

The robot model (see Figure 11) mainly consists of a 3-D mechanical structure model and 6 axis models including electrical motors, controllers and mechanical components of the axes (gear and friction). The reference trajectories for the angles and velocities of the axes are provided by a separate path planning model. The path planning is based on an algorithm that finds trajectories for the fastest movement for a given start position α to a given end position β under kinematical constraints. The constraints are defined by the maximum velocity and the maximum acceleration of the axis movements.

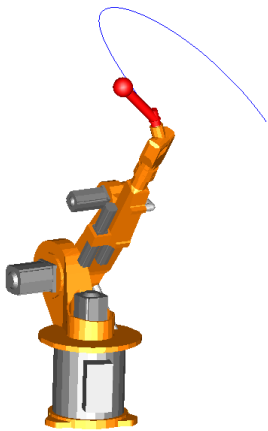


Figure 11: Animation of robot model from Modelica Standard Library.

The drawback of the path planning model is that the available maximum torque of the electrical motors is not considered. We may include them in the path planning by solving a trajectory optimization problem with the inverse dynamics model [R11] using the Optimization library. For these purposes we have to adapt the robot model. The motor, controller and friction model of the axes are removed. The rotational power train of each axes is driven by a signal based torque source. The non-causal approach of Modelica automatically leads to the inverse dynamics model when giving input signals for the robot positions [TOB01].

4.2 Trajectory Optimization Problem

The goal of the trajectory optimization problem is to find movements for the axes' angles $q(t)$. The movement from the start angles α to the end angles β should be as fast as possible under the constraints that the maximum velocity and the maximum motor torques are bounded by given values. Additionally, the angular accelerations shall be zero at the start and

the end position to avoid oscillations for the controlled robot using the computed paths as reference motion.

The mathematical formulation is as follows:

$$\begin{aligned} \min_{t_{End}, u(t)} \quad & t_{End} \quad \text{w. r. t.} \\ q(0) = \alpha, \quad & q(t_{End}) = \beta, \quad \ddot{q} = u, \\ \dot{q}(0) = \dot{q}(t_{End}) = \ddot{q}(0) = \ddot{q}(t_{End}) = 0, \\ |\dot{q}(t)| \leq v_{Max}, \quad & |\tau(t)| \leq T_{Max} \quad \text{for } t \in [0, t_{End}]. \end{aligned}$$

In our investigations we only consider the main axes 1, 2 and 3. Axes 4, 5 and 6 are fixed and do not move. Reasonable values for the maximum angular velocities v_{Max} and the maximum torques T_{Max} can be found in [OT88]. We use $v_{Max} = (3, 1.5, 5) \text{ rad/s}$ and $T_{Max} = (950, 1950, 540) \text{ Nm}$ for axis 1, 2 and 3. The adapted robot model is prepared in such a way that $q(0) = \alpha$, $\dot{q}(0) = 0$ is inherently fulfilled. The trajectory $q(t)$ is implicitly defined by B-Splines for the controls $u(t) := \ddot{q}(t)$. The trajectories for $\dot{q}(t)$ and $q(t)$ are automatically computed in the robot model by the numerical integration algorithm during the simulation of the model.

Criteria				
Description				
Criteria definitions				
Rows 19				
	name	active	usage	demand
1	"endTime"	true	Minimize	1
2	"wMax1"	true	Inequality	3
3	"wMax2"	true	Inequality	1.5
4	"wMax3"	true	Inequality	5
5	"tauMax1"	true	Inequality	950
6	"tauMax2"	true	Inequality	1900.0
7	"tauMax3"	true	Inequality	540
8	"q1"	true	e.Equality	60
9	"q2"	true	e.Equality	70
10	"q3"	true	e.Equality	35
11	"w1"	true	e.Equality	1
12	"w2"	true	e.Equality	1
13	"w3"	true	e.Equality	1
14	"a1"	true	e.Equality	1
15	"a2"	true	e.Equality	1
16	"a3"	true	e.Equality	1
17	"a01"	true	e.Equality	1
18	"a02"	true	e.Equality	1
19	"a03"	true	e.Equality	1

Figure 12: Criteria of robot path planning in Optimization setup GUI.

The trajectory optimization setup (see Figure 12) consists of three input controls $\ddot{q}(t)$ and the free parameter t_{End} . The criterion to be minimized is the end time t_{End} , whereas 6 (= 2 · 3 axes) inequality constraints are defined for \dot{q}_{Max} and τ_{Max} . The robot

model includes criteria models (see Section 3.1) to compute the absolute maxima \dot{q}_{Max} of $\dot{q}(t)$ and τ_{Max} of $\tau(t)$. There remain 12 equality constraints for $q(t_{End})$, $\dot{q}(t_{End})$, $\ddot{q}(0)$ and $\ddot{q}(t_{End})$.

The advanced feature to handle a free end time t_{End} for the trajectory optimization is implemented and will be available in the next release of the Optimization library.

4.3 Trajectory Optimization Results

We set the start trajectories for $\ddot{q}(t)$ equal to 0 and choose $t_{End} = 5$ at the beginning of the optimization. These start conditions lead to violated optimization constraints for $q(t_{End})$. The SQP algorithm succeeds in finding input functions $\ddot{q}(t)$, such that all constraints are fulfilled. Important for SQP is a high accuracy of the criteria, therefore we set the error tolerance of the integration to 10^{-12} . The error tolerance for the solution of SQP is set to 10^{-6} .

Depending on the number N of sample points for the B-splines, different solutions are found, see columns 1 and 2 in Table 3. The degree k of the polynomials is always set to $k = 3$. We tested the developed parallelization techniques (see Section 3.5) for this benchmark problem. In Table 3 the computation times for the single simulation approach are documented. Further, the speed factors using parallel simulations with 2, 4 and 8 threads are given. Since the computation of the numerical Jacobian matrix dominates the overall computation time, speed factors of pure independent simulations (compare Figure 10) can be reached for 2 and 4 parallel threads. The optimization with 8 threads is faster than using 4 threads, but the difference is smaller than in Figure 10.

Table 3: Results of the trajectory optimization with different number N of sample points for the B-splines.

N	t_{End}	Single	Parallel speed factors		
		Elapsed time	2 threads	4 threads	8 threads
5	1.60 s	30 min	1.85	2.95	3.15
8	1.48 s	150 min	1.96	3.11	3.38
10	1.42 s	251 min	1.95	3.12	3.34
20	1.40 s	908 min	2.00	3.33	3.58
30	1.40 s	1228 min	2.02	3.37	3.68

Figure 13 illustrates the solutions $q_2(t)$, $\dot{q}_2(t)$ and $\ddot{q}_2(t)$ for $N = 5, 8, 10$ and 20. It is obvious, that the

velocity constraint $\dot{q}_2 \leq v_2 = 1.5 \text{ rad/s}$ is an active constraint. In Figure 14 it can also be seen, that the motor torque is inside the demanded ranges. The trajectory for the torque of axis 3 hits the border lines several times.

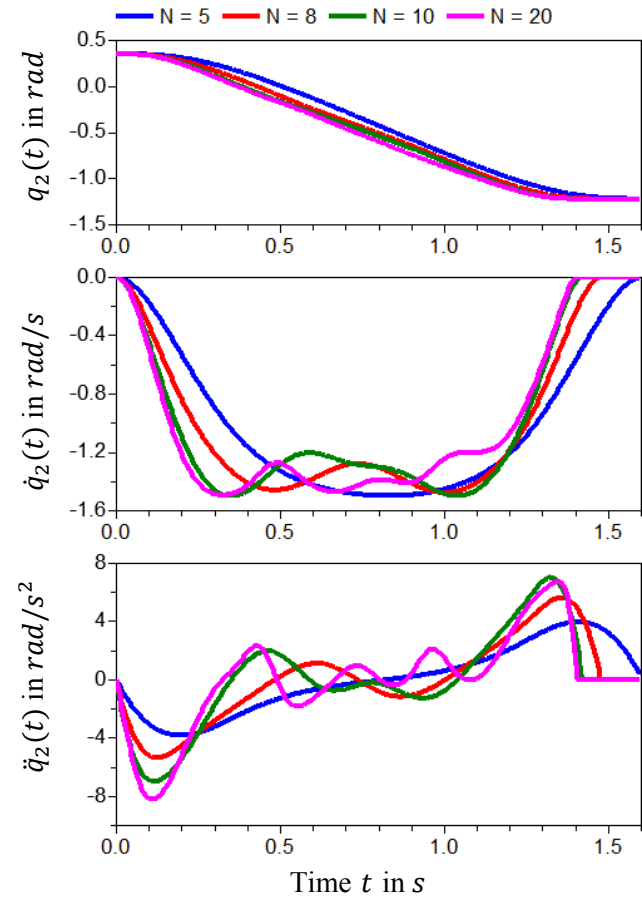


Figure 13: Result trajectories with different number N of sample points for the B-splines.

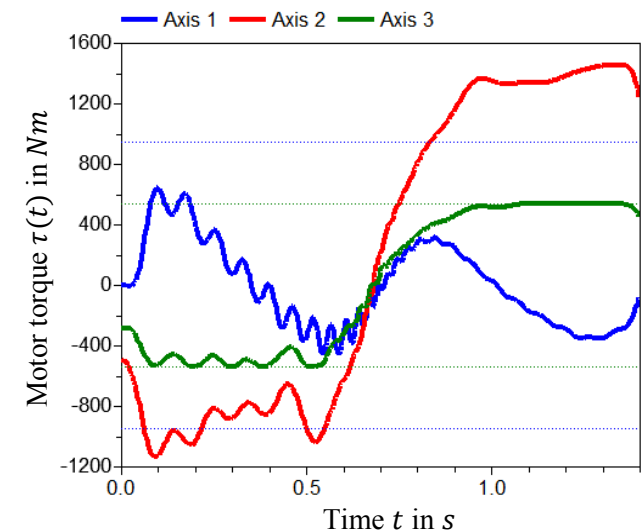


Figure 14: Motor torque for different axes. The optimization solution is computed with $N = 20$ sample points.

5 Conclusions

A library for solving interactive optimization tasks is presented. Both function and different model based optimization tasks are available to support the engineer in improving his system design by sophisticated numerical optimization algorithms. Additionally, optimization runs may be accelerated by automated parallel model simulations on multi-core machines. Version 2.1 of the Optimization library is available along with the release of Dymola 2013.

6 Acknowledgement

The support of H.-D. Joos, M. Otter, M. Reiner and K. Schnepfer (all members of DLR Institute of System Dynamics and Control) in developing the Optimization library and the application example is gratefully appreciated. Improvements of Dymola by Dassault Systèmes AB to support the Optimization library are acknowledged. Partial financial support of DLR by BMBF (BMBF Förderkennzeichen: 01IS07022F) for this work within the ITEA2 project EUROSYSLIB [E12] is highly appreciated. Also, the constructive comments of the anonymous paper reviewers are appreciated.

References

- [AAG+10] Åkesson J., Årzén K.-E., Gäfvert M., Bergdahl T. and Tummescheit H.: *Modeling and Optimization with Optimica and JModelica.org – Languages and Tools for Solving Large-Scale Dynamic Optimization Problems*. Computers and Chemical Engineering, Vol. 34, Issue 11, pp. 1737-1749, 2010.
- [B01] Betts J. T.: *Practical Methods for Optimal Control Using Nonlinear Programming*. SIAM Press, Philadelphia, Pennsylvania, USA, 2001.
- [CJP08] Chapman B., Jost G. and van der Pas R.: *Using OpenMP, Portable Shared Memory Parallel Programming*. The MIT Press, Cambridge, Massachusetts, London, England, 2008.
- [DH02] Deufflhard P. and Hohmann A.: *Numerische Mathematik I. Eine algorithmisch orientierte Einführung*. 3. Auflage, de Gruyter, Berlin, Germany, 2002.
- [DS12a] Dassault Systèmes AB: *CATIA*. www.3ds.com/products/catia.
- [DS12b] Dassault Systèmes AB: *Dymola*. www.dymola.com.
- [E05] Ehrgott M.: *Multicriteria Optimization*. Second Edition, Springer, Berlin, Heidelberg, Germany, 2005.
- [E12] EUROSYSLIB, ITEA2 06020, www.eurosyslib.com.
- [EOM+05] Elmqvist H., Olsson H., Mattsson S. E., Brück D., Schweiger C., Joos D. and Otter M.: *Optimization for Design and Parameter Estimation*. Proc. of 4th International Modelica Conference, pp. 255-266, Hamburg, Germany, 2005.
- [J11] Joos H.-D.: *MOPS - Multi-Objective Parameter Synthesis, User's Guide V6.2*. Institute of Robotics and Mechatronics, DLR Oberpfaffenhofen, Germany, 2011.
- [JBL+02] Joos H.-D., Bals J., Looye G., Schnepfer K. and Varga A.: *A Multi-Objective Optimisation based Software Environment for Control System Design*. Proc. IEEE International Conference on Control Applications, pp. 7-14, Glasgow, Scotland, Sept. 18-20, 2002.
- [K10] Köppern J.: *Integrierte Fahrzeugregelung durch einen hybriden Ansatz aus inversem Modell und modellprädiktiver Optimierung*. GMA-Fachausschuss 1.40 "Theoretische Verfahren der Regelungstechnik", Salzburg, Austria, 2010.
- [LAS97] Lewis A, Abramson D. and Simpson R., *Parallel non-linear optimization: Towards the design of a decision support system for air quality management*. Proc. of IEEE Supercomputing 97, San Jose, USA, 1997.
- [OT88] Otter M. and Türk S.: *The DFVLR Models 1 and 2 of the Manutec r3 Robot*. DFVLR-Mitteilung 88-13, Institut für Dynamik der Flugsysteme, DLR Oberpfaffenhofen, Germany, 1988.
- [R11] Reiner M.: *Modellierung und Steuerung von strukturelastischen Robotern*. Ph.D. thesis, University of Technology, Munich, 2011.
- [TOB01] Thümmel M., Otter M. and Bals J.: *Control of Robots with Elastic Joints based on Automatic Generation of Inverse Dynamics Models*. Proc. of IROS, pp. 925-930, Maui, Hawaii, USA, 2001.
- [TNT+11] Thieriot H., Nemer M., Torabzadeh-Tari M., Fritzon P., Singh R. and Kocherry J. J.: *Towards Design Optimization with OpenModelica Emphasizing Parameter Optimization with Genetic Algorithms*. Proc. of 8th International Modelica Conference, pp. 756-762, Dresden, Germany, 2011.

A Planar Mechanical Library for Teaching Modelica

Dirk Zimmer

Deutsches Zentrum für Luft- und Raumfahrt (DLR)
Münchner Strasse 20, 82234 Weßling, Germany
dirk.zimmer@dlr.de

Abstract

Teaching Modelica to students of a university requires suitable example models. This paper describes a planar mechanical library that is primarily conceived for didactical purposes. It is simple, built out of a few components only, but it enables the modeling of interesting and complex systems. The library is freely available and supported by various Modelica environments.

Keywords: Education; Planar Mechanics;

1 Introduction

1.1 Motivation

This paper presents a planar mechanical library that has been primarily designed for didactical purposes. The idea of such a library is that it is simple and easy to understand. In this way, the students can focus on learning the principles of equation-based modeling and they can avoid the lot of peculiar particularities that have meanwhile become part of the language.

We have used this library in the Modelica course at the technical university in Munich [8]. The course is enlisted in the computer science department. The students of this class mostly study computer science, applied mathematics or physics. Computer science students in Munich do not have any physics course in their basic curriculum. Hence, explaining the modeling of physical systems requires explaining the physics as well, in this particular case: the fundamental laws of motion.

In planar mechanical systems, we describe the physics of a multi body system in a two-dimensional plane. Each body position can be described by the coordinates x and y and its orientation by the angle φ (see Figure 1). Each body has a mass and its inertia can be described by a single scalar.

Planar models of mechanical systems are useful for a number of applications. Very popular is their use for contact problems that are a lot simpler in 2D

than in 3D. The modeling of gear wheel interaction is one such example [5]. For this paper their use in teaching is of course the main issue.

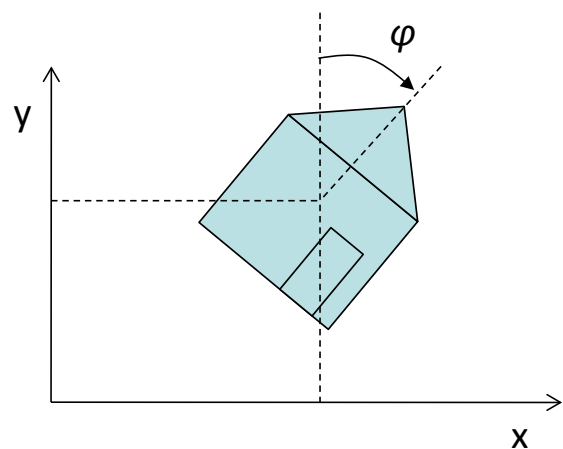


Figure 1: Representation of an object in planar space

1.2 Suitability of planar mechanics

Planar mechanical systems are ideally suited for teaching equation-based modeling, because their components are easy to model and to understand but the resulting systems are often complex in behavior and demanding in their computational aspects. Or to put it in short terms: you can do a lot of cool stuff by simple means.

From the modeling side, planar mechanics offers the following advantages:

- Planar mechanical systems are tangible and visual systems. All students have played with mechanical systems before in their life and everyone has an intuitive (and sometimes wrong) understanding about their motion. This motion can be visualized in an animation, which is more appealing to students than studying plots.
- The physical laws of planar mechanical systems are basically taught already in high-school. D'Alembert Principle and Newton's Law look familiar to the students. The equations of motion themselves are relatively easy.

- Planar mechanical systems can be steered either by human interaction or by a control law. Again these tasks are very tangible and concrete: everyone has steered a bicycle and everyone has tried to balance a pen in his life.

The resulting system can then be used to demonstrate and study the advantages and difficulties of equation-based modeling.

- First of all, mechanical systems require true non-causal equation-based modeling. Modeling methods that are based on the computational flow such as Simulink are of very limited use in this domain. A kinematic loop can be used as an illustration.
- Also kinematic loops require the solution of non-linear equation systems. The corresponding examples can be used to explain techniques for initialization and state selection.

In contrast to planar mechanical systems, 1D and 3D mechanical system are not so well suited for teaching.

1D mechanical systems are too simple. Of course, We teach both rotational and translational mechanic prior to planar system, but many interesting configurations such as kinematic loops do not naturally exist in 1D. Hence, the topic does not bear long and quickly gets boring unless you enter the specifics of drive-train modeling which is misplaced in a general Modelica course.

3D mechanical systems on the other side are way too complex. A short look on the components of the standard MultiBody Library [2,7] makes this clear. In 3D, the description of a body orientation can be performed in many different and potentially redundant ways. This redundancy then leads to further difficulties so that kinematic loops require special treatment. In planar mechanics, the orientation is uniquely described by a single angle and kinematic loops do not require special modeling tools.

2 State of the Art in planar mechanical modeling

The library presented in this paper is not the first planar mechanical library that has been developed in Modelica.

Indeed, we have developed one of the first variants as part of the MultiBondLib [7]. It is freely available and it is also well suited for teaching but only in a course where bondgraphic modeling is part

of the program. In contrast, the new library is directly based on equations and does not require the knowledge of bondgraphs. Furthermore, because of the use of bondgraphs in the MultiBondLib the connectors contained redundant information and kinematic loops required special handling.

A second planar library has been developed by Hübinger and Otter [4]. In addition to the basic mechanical components (joints and body parts), the library contained models for the contact of curved surfaces. Although, it was envisioned that this library becomes part of the Modelica Standard Library (MSL), this has not yet taken place.

Furthermore new planar mechanical elements have developed by van der Linden [5] for the modeling of gearwheels. This developments use the same interfaces and components as the planar mechanical library presented here.

2.1 Contributions of this Paper

Since already a significant amount of effort has been spent on the development of Modelica code for planar mechanics, it is important to clarify the contribution of this paper. Essentially there are three major objectives for this work:

- **Presentation of a didactical library:** This is the major part of this paper (section 3 to 5). I will present the interfaces and the structure of the library and show how simple the individual components can be modeled.
- **Cross-Platform Library for different compilers:** The ability to compose complex systems out of simple components using only a smaller subset of the language is not only interesting for students but also for compiler developers. The library turns out to be very well suited for testing the abilities of various Modelica environments. Also for teaching purposes, it is good if the material is not bounded to a certain software tool but of general applicability. More on this topic in section 6
- **Establishment of a standard interface for planar mechanics:** The planar mechanical library for didactical purposes is not supposed to become part of the MSL. Libraries that are part of the MSL must be optimized with respect to usability. This in part conflicts with desired level of simplicity for teaching. However, there is no reason why a potential library for planar mechanics in the MSL and the didactical library should use different interfaces.

3 Structure of the library

The interface of a planar mechanical component represents a flange point. This point is determined by a fixed position in the plane (x, y) and a fixed orientation angle (ϕ). Forces in x and y direction (f_x, f_y) as well as a torque (τ) may act on the flange point. The corresponding Modelica connector is hence designed as follows:

Listing 1: Connector code

```
connector Frame
"General Connector for planar mechanical components"

SI.Position x "x-position";
SI.Position y "y-position";
SI.Angle phi "angle (counter-clockwise)";
flow SI.Force fx "force in x-direction";
flow SI.Force fy "force in y-direction";
flow SI.Torque t "torque (clockwise)";

end Frame;
```

For simplicity, the potential use of vectors in the connector has been omitted. For beginners it is a little easier, to work with x, y , and ϕ than with a vector $r[2]$ and ϕ . The same holds for the forces. Given this connector, a variety of planar mechanical components can be implemented. Figure 2 provides an overview of the library content.

The standard components are parts and joints. These elements were designed in strong resemblance to their counterparts in the Modelica MultiBody library. In addition to the standard components, the library contains sub-packages for vehicle wheels and gearwheels.

The wheel models can be used to move with a wheel on the x, y -plane. There are ideal wheel models and simple slip based models inspired by previous works [6].

Future versions of this library may also contain the gear wheel models out of the work of van der Linden [5]. They can for instance be used to assemble a planetary gear box.

All elements in this library contain a suitable visual representation for the animation. For simplicity though, the animation is not as configurable as in the MultiBody library. Another difference to the MultiBody library is that there is no World model available in this library. Again the sheer simplicity is preferred over a more elaborate solution.

The library features a large set of examples that demonstrate the variety of systems that can be as-

sembled from these components: pendulum, crane crab, kinematic loops, or even two-track car vehicle models are included. Also examples of controlled systems and model inversion are contained in this library.

The library itself is available at [8] or at the Mod- elica Website. This is made publicly available and represents the standard version. The examples in this version are all suitable for testing purposes. Further- more this library is self-contained only requiring a few elements of the standard library but not requiring any other library.

The planar mechanical library that is being used in the lecture course is slightly different. First of all it is developed in several steps as the course pro- ceeds. In its latter stages, it also contains elements from DLR libraries. The lecture course contains also slides explaining the components of this library at great level of detail.

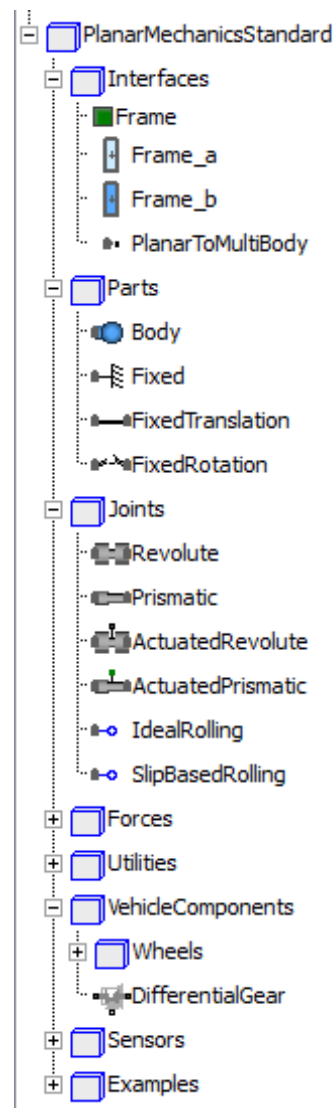


Figure 2: Structure of the planar mechanical library

4 Teaching Modelica

4.1 Context

When the library is used for teaching, it is not presented as a whole but gradually developed together with the students. The goal is that the students learn all relevant processes of modeling in Modelica: from punching in equations, plugging together components to designing a whole library.

In the course “Virtual Physics”, the library is being used from lesson 5 on. In the first 4 lessons, the students learn the basics of equation-based modeling and the Modelica language. After going through examples of 1D mechanical systems, we start by the most basic mechanic components.

4.2 Component Modeling

The most important component is of course the body component:

Listing 2: Body component

```

model Body "Body component with mass and inertia"

  Interfaces.Frame_a frame_a;

  parameter SI.Mass m "mass of the body";
  parameter SI.Inertia I "Inertia of the Body";
  parameter SI.Acceleration gx =0
    "gravity acceleration (in x) acting on the mass";
  parameter SI.Acceleration gy=-9.81
    "gravity acceleration(in y) acting on the mass";

  SI.Velocity vx "velocity in x";
  SI.Velocity vy "velocity in y";
  SI.AngularVelocity w "angular velocity";
  SI.Acceleration ax "acceleration in x";
  SI.Acceleration ay "acceleration in y";
  SI.AngularAcceleration z "angular acceleration";

equation
  //The velocity is a time-derivative of the position
  vx = der(frame_a.x);
  vy = der(frame_a.y);
  w = der(frame_a.phi);

  //The acceleration is a time-derivative of the velocity
  ax = der(vx);
  ay = der(vy);
  z = der(w);

  //Newton's law
  fx + m*gx = m*ax;
  fy + m*gy = m*ay;
  frame_a.t = I*z;

end Body;

```

Even with plenty of comments the code remains compact and is very easy to understand. For the first version, everything that may distract the student has been removed. Gravity acceleration is a simple parameter and does not be read out of a strange “world model”. There is no animation and there are no options for initialization or state-selection that pollute the code. Just the bare physical equations form the model.

In this version, also no vector notation is used. For students of a technical university it seems to cause no problems in understanding the model code. Teaching experience from universities of applied sciences indicates that vector notation is better introduced later on. Vector notation is used in a subsequent version, where also the code of the animation is added. The students know at this stage that this code is non-essential.

For joint elements, a neutral element is a good starting point. This element implements the lever principle but exhibits no forces on its connectors.

Listing 3: Neutral component

```

model Neutral
  //This component has two frames...
  Interfaces.Frame_a frame_a;
  Interfaces.Frame_a frame_b;

equation
  //...but exhibits no effect.
  frame_a.fx = 0;
  frame_a.fy = 0;
  frame_a.t = 0;

  //This is the balance of force and torque
  including the lever principle
  frame_a.fx + frame_b.fx = 0;
  frame_a.fy + frame_b.fy = 0;
  frame_a.t
  + frame_b.t
  + (frame_b.x - frame_a.x)*frame_b.fy
  - (frame_b.y - frame_a.y)*frame_b.fx
  = 0;

end Neutral

```

Any joint can now be implemented by replacing the assignment of zero force with the corresponding positional constraints. Furthermore, the lever principle can often be simplified. Let us for instance look at the revolute joint. Here, two positional constraints are enforced: the position must be equal in direction of x and y. Since there is no distance between the two frames, the lever principle degenerates to a balance of torque.

Listing 4: Revolute joint, first version

```

model Revolute
  Interfaces.Frame_a frame_a;
  Interfaces.Frame_a frame_b;

equation

  //frame_a.fx = 0 gets replaced by
  frame_a.x = frame_b.x;

  //frame_a.fy = 0 gets replaced by
  frame_a.y = frame_b.y;

  frame_a.t = 0;

  //since there is no difference in position
  //the lever principle can be simplified
  frame_a.fx + frame_b.fx = 0;
  frame_a.fy + frame_b.fy = 0;
  frame_a.t + frame_b.t = 0;

end Revolute;

```

In a second version, two differential equations and one algebraic equation are added since the joint is well suited to describe the motion of the system.

Listing 5: Revolute joint, second version

```

model Revolute
  Interfaces.Frame_a frame_a;
  Interfaces.Frame_a frame_b;

  //These 3 variables help to describe the motion of a system
  SI.Angle phi
  SI.AngularVelocity w;
  SI.AngularAcceleration z;

equation

  //For 3 more variables we need 3 more equations:
  frame_a.phi + phi = frame_b.phi;
  w = der(phi);
  z = der(w);

  //Known material...
  frame_a.x = frame_b.x;
  frame_a.y = frame_b.y;
  frame_a.t = 0;

  frame_a.fx + frame_b.fx = 0;
  frame_a.fy + frame_b.fy = 0;
  frame_a.t + frame_b.t = 0;

end Revolute;

```

In this way, also a fixed translation element can be explained. The prismatic joint can then be presented as a translational element of variable length.

4.3 Valuable Examples for Teaching

Having available only five component models for

- body with mass and inertia,
- revolute joint,
- prismatic joint,
- fixed translation,
- and global fixation

enables us to compose already a lot of interesting models.

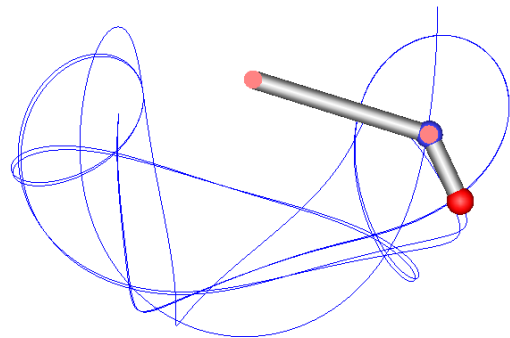


Figure 3: Chaotic trajectory of a double pendulum

The famous double pendulum can be used to demonstrate chaotic system behavior. Figure 3 shows the erratic trajectory of the peak of the pendulum. Simulating with different values for precision yields each time a completely new trajectory and no convergence can be reached. The students learn the important lesson that a simple non-linearity can lead to totally unpredictable and chaotic systems.

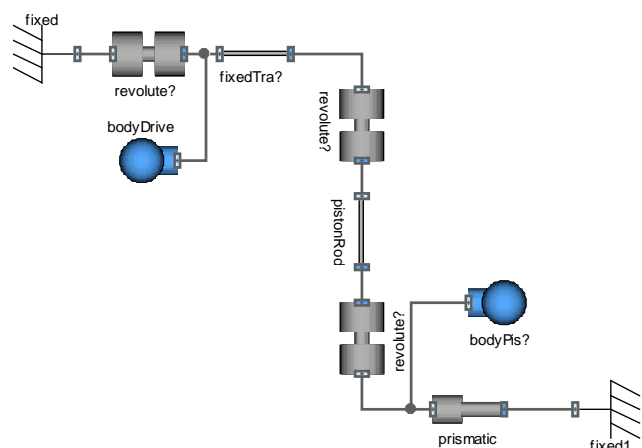


Figure 4: Model diagram of a simple piston engine

Figure 4 displays the model diagram of a piston engine. It represents a kinematic loop: although there are four joint elements, the complete system has just one degree of freedom. This example is used to ex-

plain the mechanism of initialization and state selection to the students. The joint elements are then further enhanced by an initialization section and attributes for state selection. Furthermore, the students learn about the Pantelides algorithm for reducing the differential index of a system.

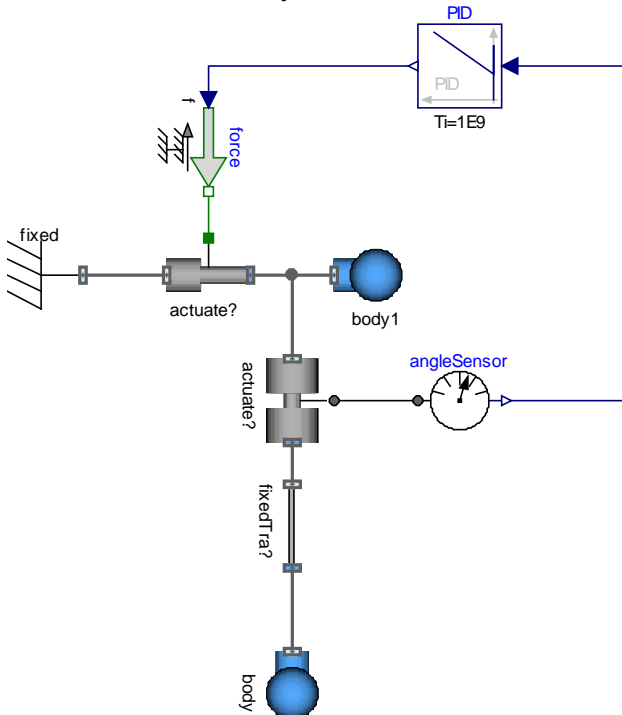


Figure 5: Model diagram of an inverted pendulum controlled by a PID element

The inverted pendulum is a famous example in control theory. It is easy to model by using the planar mechanical components. A simple PID controller can be added to show how a controller can be designed in Modelica. Furthermore it is possible to invert the model by stipulating the trajectory and computing the forces. In this way, the students can learn how flexible a Modelica model can be used: not only for simulation but also for control design and model inversion.

5 Tire and vehicle models

Whereas the standard components already enable the creation of many interesting examples, planar mechanical systems can also be used to model vehicles driving on the plane. To this end three separate wheel models are provided:

- An ideal rolling wheel
- A dry-friction based wheel
- A slip-based wheel

Listing 6 presents the code for the ideal rolling wheel. Although being already significantly more complex, this component is not beyond what a good student can learn to understand if he is supported by sufficient explanations and further material.

Listing 6: Ideal wheel

```

model IdealWheelJoint

  Interfaces.Frame_a frame_a;
  Rotational.Interfaces.Flange_a flange_a;

  parameter SI.Length radius
    "radius of the wheel";
  parameter SI.Length r[2]
    "driving direction of the wheel at angle phi = 0";
  final parameter SI.Length l = sqrt(r*r);
  final parameter Real e[2] = r/l
    "normalized driving direction";

  Real e0[2] "normalized direction w.r.t inertial system";
  Real R[2,2] "Rotation Matrix";

  SI.AngularVelocity w_roll "roll velocity";
  SI.Velocity v[2] "transl. velocity";
  SI.Velocity v_long "velocity in longit. direction";
  SI.Acceleration a "accel. of driving velocity";
  SI.Force f_long "longitudinal force";

  equation

    //Resolve the normalized driving direction in the
    //inertial coordinate system
    R = {{cos(frame_a.phi), -sin(frame_a.phi)},
         {sin(frame_a.phi), cos(frame_a.phi)}};
    e0 = R*e;

    //Project the longitudinal velocity in the planar space
    // (this implies that the lateral velocity is zero)
    v = der({frame_a.x, frame_a.y});
    v = v_long*e0;

    //Implement the law of ideal rolling
    w_roll = der(flange_a.phi);
    v_long = radius*w_roll;
    a = der(v_long);

    //Project the force on the longitudinal direction
    {frame_a.fx, frame_a.fy}*e0 = f_long;

    //model the drive torque
    -f_long*radius = flange_a.tau;

    //There is no bore torque
    frame_a.t = 0;

  end IdealWheelJoint;

```

The code for the other two wheel models is only a little more complex. The students have to learn about friction characteristics and regularization techniques. Given these wheel models, a simple one-track car model can be composed in five minutes:

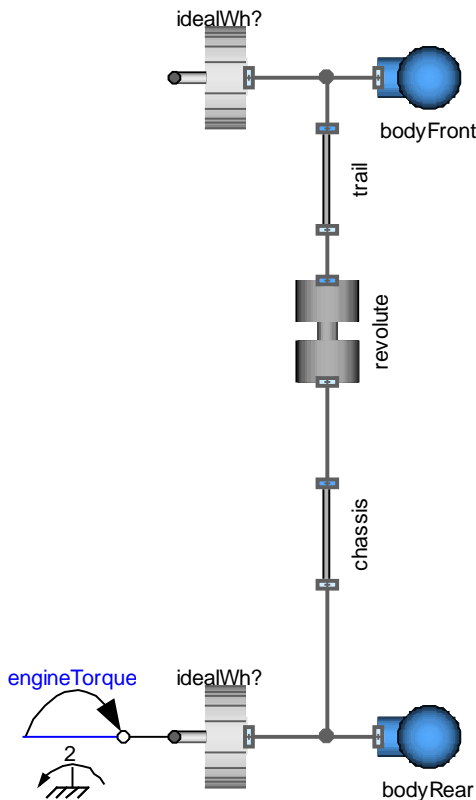


Figure 6: Model diagram of a simple one track vehicle

Such a model is sufficient to study the influence of the trail or the basic difference between front-wheel drive and rear-wheel drive.

The highlight of the course is a two-track car model with slip-based wheels. It is enhanced by a simple 3D chassis that computes the load balance on the four wheels. The car model can be simulated in real-time. It is also visualized in real time by the use of the SimVis Library [1] (see Figure 7) and can be controlled online by the keyboard using components from the Modelica Device Drivers library [3]. As a result, the students can drive their own car model in 3D just as in a computer game. Such an example attracts many students to the course and helps to keep up their motivation during the course.

6 Cross-platform compatibility

Since the library uses only a subset of the Modelica language that consists entirely out of well-established language constructs, it can be supported by a large set of different Modelica compilers al-

ready now. 15 examples have been selected for testing the results of various Modelica simulation environments. The current test results are summarized in figure 8. It shows the test results for all 17 examples and for for different compilers.

First of all, Dymola[9] offers full support of the library. It is also the environment that has been used for the development of the library and that I use for teaching.

JModelica[11] is also able to parse and process the entire library. It does not offer dynamic state-selection as in Dymola but this feature is not so essential for a didactical library.

OpenModelica[10] can also parse the entire library. The correct translation and simulation is possible for large set of examples but not for all of them. In some more complex examples, the back-end of the compiler still has some problems with the non-holonomic constraints equations that originate from ideal rolling parts.

Also SimulationX[12] offers almost full support of the library. Some examples require a non-standard solver but these are this was the only small problem that occurred. For one example of a kinematic loop, SimulationX started with the wrong initial position but this might be due to modeling ambiguity.

In all cases the compiler developers are working on the occurring problems and there is a fair chance that a complete support of the library can be realized soon.

Test of MapleSim[13] have not yet been completed. First results indicate that MapleSim parses the code correctly and that the simulator is capable of simulating the test cases. The current problems concern the usability of the models but these problems should be solved for the new version of MapleSim.

Tests within Wolfram SystemModeler [14] have not yet been done.



Figure 7: 3D-Realtime visualization of the two track vehicle

	Name	Dymola	Open Modelica	JModelica	SimulationX
1	FreeBody	OK	OK	OK	OK
2	Pendulum	OK	OK	OK	OK
3	DoublePendulum	OK	OK	OK	OK
4	CounterSpin	OK	OK	OK	OK
5	CraneCrab	OK	OK	OK	OK
6	CraneCrabControlled	OK	OK	OK	OK
7	InvertedCraneCrab	OK	OK	OK	OK
8	WheelBasedCraneCrab	OK	OK	OK	OK
9	PistonEngine	OK	OK	OK	OK
10	KinematicLoop	OK	F?	OK	F?
11	TestIdealWheel	OK	F	OK	OK
12	TestDryFrictionWheel	OK	OK	OK	OK
13	TestSlipBasedWheel	OK	OK	OK	OK
14	SingleTrack	OK	F	OK	OK
15	TwoTrack	OK	OK	OK	OK

Categories **OK** Runs successfully
 F Fails

Figure 8: This table displays the current support of the library among different Modelica environments

7 Conclusions

Ultimately, the goal is to have a didactical library available that can be used to teach Modelica in different modeling and simulation environments.

I personally hope that this library helps other lecturers to create their Modelica courses. It can be used for free under the Modelica 2 license. Suggestion (or even better: contributions) that help to improve the quality of the library are always highly welcome.

Acknowledgements

I would like to acknowledge the effort of Thomas Schmitt and Markus Andres who gathered further teaching experience using planar mechanical models at the University of Applied Sciences Vorarlberg.

I would like to thank Franciscus van der Linden for his contribution to the standardization of the interface and the gear wheel models in the library.

Many thanks to Francesco Casella for having the idea to use this library as test-case for different compilers. Johan Åkesson, Jens Frenkel and Adrian Pop helped in testing the library on OpenModelica and JModelica.

Thanks for Ingrid Bausch-Gall and Jakob Tobolar for the help with SimulationX and thanks to Matthias Reiner for a first investigation in MapleSim.

References

- [1] Bellmann, Tobias (2009) Interactive Simulations and advanced Visualization with Modelica. Proc. of the 7th International Modelica Conference, 20.-22. Sept. 2009 , Como, Italy
- [2] Cellier, F.E. and D. Zimmer (2006), *Wrapping Multi-bond Graphs: A Structured Approach to Modeling Complex Multi-body Dynamics*, keynote presentation, 20th European Conference on Modeling and Simulation, Bonn, Germany, May 29-31, 2006
- [3] Elmquist, H., et. al. (2009) Modelica for embedded systems. Proc. of the 7th International Modelica Conference, 20.-22. Sept. 2009 , Como, Italy
- [4] Höbinger, M. and M. Otter (2008), *Planar-MultiBody - A Modelica Library for Planar Multi-Body Systems*. Proc. 6th International Modelica Conference, Bielefeld, Germany
- [5] van der Linden, F. (2012), *Modelling of Elastic Gearboxes Using a Generalized Gear Contact Model*. In review for the Proc. of the 9th Modelica Conference, Munich, Germany
- [6] Zimmer, D. and M. Otter (2010), Real-Time Models for Wheels and Tires in an Object-Oriented Modelling Framework. *Journal of Vehicle System Dynamics*. Volume 48, Issue 2, pp. 189-216 .

- [7] Zimmer, D. and F.E. Cellier (2007), The Modelica Multi-bond Graph Library, *Simulation News Europe*, Volume 17, No. 3/4, pp. 5-13.
- [8] Zimmer D. *Virtual Physics*. Lecture Notes available at: www.robotic.dlr.de/dirk.zimmer

Tool References

- [9] Dymola:
www.3ds.com/products/catia/portfolio/
- [10] OpenModelica:
www.openmodelica.org
- [11] JModelica:
www.jmodelica.org
- [12] SimulationX:
www.itisim.com
- [13] MapleSim:
www.maplesoft.com/products/maplesim
- [14] Wolfram SystemModeler:
<http://www.wolfram.com/system-modeler/>

DyMoRail: A Modelica Library for modelling railway buffers

Elisabeth Dumont Werner Maurer

Zentrum für angewandte Mathematik und Physik, Zürcher Hochschule für Angewandte Wissenschaften
Technikumstrasse 9, Winterthur, 8401 Switzerland

Abstract

This article gives an overview of the DyMoRail library. The aim of this Modelica library is the simulation of longitudinal dynamics of entire railway trains. The DyMoRail library allows an efficient simulation of complete train compositions in various configurations. The library contains different car models, buffers, couplers equipped with both friction and elastomer springs, as well as the center-buffers for multiple units. DyMoRail allows to simulate the entire motion cycle that the buffer undergoes during a collision. The robust programming of the basic models allows simulations for arbitrary combination of buffers, couplers and destruction tubes. Different modelling techniques (SIMULINK, STELLA) have been explored. Since the modular structure of Modelica allows fast and simple setup of models including different types of rolling stock and different types of couplers and buffers, it was decided to build this library in Modelica. This simulation environment was successfully used by Schwab Verkehrstechnik AG during the development of their state-of-the-art center coupler product family. Within DyMoRail2 we intend to implement further features and improve the modularity and flexibility of the library.

Keywords: library, mechanics, railway

1 Introduction

Buffers and couplers are an essential part of the railway wagon. They have to be optimized for new wagon types to work for different train compositions. They have to absorb minor impacts, take up slack between locomotive and wagons and bear the load of preceding wagons when pushing. Years ago it was good enough for couplers and buffers to fulfil UIC (International Union of Railways) standards. But nowadays manufacturers only survive in this competitive market if they are able to offer optimized solutions regarding force, energy absorption, and driving comfort. Mod-

elling plays an important role in this optimization process. One of the main requirements to this rail model are that it should allow easy substitution of components and handling of different combinations of subsystem parts.

Schwab Verkehrstechnik AG and ZHAW carried out a project funded by CTI (Swiss Federal Commission for Technology and Innovation) to develop a simulation tool which allows to model longitudinal dynamics of entire railway trains. During the following years a Modelica library has been developed which is called DyMoRail. The DyMoRail library allows an efficient simulation of complete train compositions in various configurations. The library contains a number of different car models, buffers, couplers equipped both with friction and elastomer springs, as well as the center-buffers for multiple units (such as Seetalbahn, Turbo, Flirt). DyMoRail allows to simulate the entire motion cycle during a collision (retraction of the buffer, force increase with stroke of the buffer, extension of the buffer, and finally the separation of the wagons).

The robust programming of the basic models allows for arbitrary combination of buffers, train draw rod and destruction tubes. In a first attempt, simulations were performed with SIMULINK. But it turned out that in SIMULINK a completely new model had to be programmed from scratch for each combination. Therefore Schwab Verkehrstechnik and ZHAW decided to build a new library based on Modelica. The modular structure of Modelica allows fast modifications of the model by simple replacement of entire subsystems. In this paper we will present the structure of the existing library, show some examples and propose some improvements that will lead to a new version DyMoRail2, which will be constructed in collaboration with Schwab Verkehrstechnik AG and is funded by CTI.

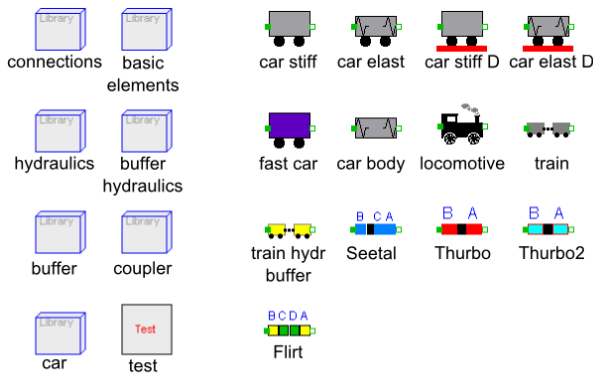


Figure 1: DyMoRail library structure

2 Library Structure

The DyMoRail library structure is shown in Figure 1. The fundamental packages and models are explained in the following paragraph. The library consists of seven sublibraries: connectors, basic elements, hydraulics, buffer hydraulics, buffers, couplers and cars.

2.1 Connectors

The sublibrary connectors contains the mechanical and hydraulic connections as well as the connections between the hydraulic buffers and the respective buffer hydraulics.

2.2 Basic Elements

This sublibrary contains different models of springs and buffers, as well as friction between car and railway track. The submodels "buffer bush" and "coupler bush" simulate the fundamental behaviour of the buffers and couplers. Four different operation modes of the buffer bush are distinguished: free, pretension, deformation and arrested. In free mode, the buffer plates do not touch and the force is zero. In the pretension mode the force increases. In the deformation mode the buffer spring and buffer hydraulics are loaded. In the arrested mode the force increases steeply. The additional state forward, backward and halt describe the actual condition of the bush. In addition friction is also modelled in the bush.

2.3 Hydraulics

In addition to the basic elements for viscous flow, this sublibrary contains hydraulic accumulator and check valves and multiplier valves for various buffers and couplers. Each multiplier valve has three signal inputs:

The first signal yields the state of the buffer. The valve opens only if the buffer state is on deformation and is not retracted. The second input provides the opening of the aperture so that oil can flow into the hydraulic buffer. The third input provides the deformation of the buffer bush.

2.4 Buffer and Couplers

These sublibraries contain products of the company Schwab Verkehrstechnik AG, such as buffers, couplers, coupling rods and railway compositions. Elastomer springs are commonly used, because they are cost saving and robust. They show a non linear characteristic and have high inner friction. The friction depends on the buffer force and has both a linear and a non-linear part.

The library contains a basic model for both the hydraulic buffer and coupler. The hydraulics, which have to be reconfigured for each train, are filed in the buffer hydraulics sublibrary.

2.5 Cars

This sublibrary (also shown in Figure 1) contains elements, which allow the modelling of cars as rigid or flexible bodies, as well as car bodies and locomotives. Two additional models allow to model freight trains with an arbitrary number of wagons. These trains are equipped with either standard buffers or hydraulic buffers. The library contains further models which describe multiple units of Stadler Rail AG (such as GTW, FLIRT, KISS).

3 Examples

Two different examples are presented in the following paragraph.

3.1 1 g-Buffer

The acceleration of lightly loaded freight cars during a shunting impact can reach levels as high as 40 m/s^2 (4 g). Such hard collisions mean a high risk of damage to the freight. Since for transportation by trucks much lower accelerations of the order of 0.8 g occur, this is a severe disadvantage of the rail transport compared with road transport. The 1 g-buffer was developed in order to protect damageable freight during shunting impacts. This buffer should keep the maximum acceleration of the wagons below 10 m/s^2 at an impact velocity of 7.2 km/h. The buffer shows the same static

behaviour as a conventional UIC–526 buffer, i.e. the force increases up to a value of 900 kN at a stroke of 150 mm. Under static load, the buffer can only retract by half of its length up to a maximum force of 150 kN. Due to an elaborate valve control the 1 g-buffer is dynamically more flexible than statically. At first this buffer has been modelled with SIMULINK. However, modelling with DyMoRail allows a larger variety of different scenarios. Besides the DyMoRail model is much more precise and detailed.

In Figure 2 a model for a collision between two wagons is depicted. A freight car of 80 t, respectively 30 t, collides with a car at rest. The moving car is equipped with a 1 g-buffer and the car at rest with standard UIC-buffers. Figure 3 and 4 show the simulation results. The force-stroke-behaviour of the 1 g-buffer is drawn during shunting impacts. For both cases, the acceleration of the cars does not exceed 10 m/s^2 (1 g). 1 g buffers are used nowadays mostly for freight cars that transport road semi-trailers. According to the simulated data the 1 g buffer complies with DB Cargo standards.

The model has been validated with measurements performed on the 1 g buffer[1].

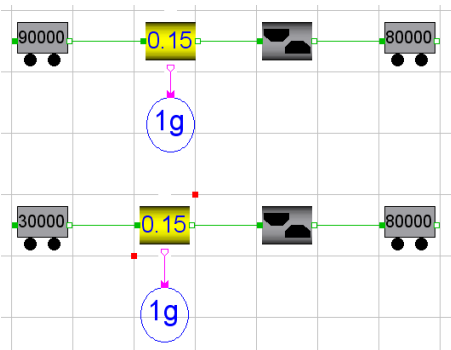


Figure 2: Model for a collision between two wagons. A freight car of 90 t, or 30 t respectively, collides with a car at rest. The moving car is equipped with a 1 g-buffer (yellow) and the car at rest with standard UIC-buffers (grey).

3.2 S-Bahn

For Zurich S-Bahn trains of the third generation, multiple units consisting of 6 double decker coaches are used. A single assembly has a mass of 312 t, a total length of 150 m and can take up to 1694 passengers. It is obvious that, during shunting, the rolling stock must not be damaged at all. This means that the central coupler must not be damaged during a collision of such a

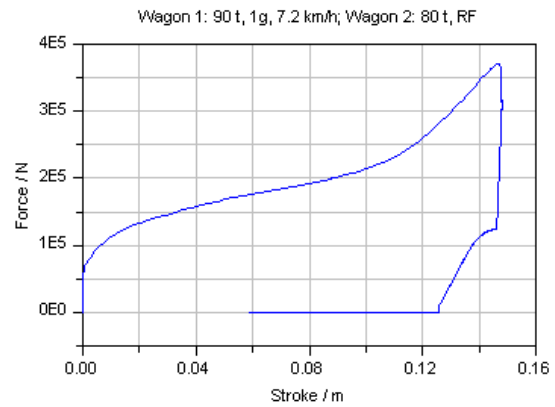


Figure 3: Force-stroke diagramm of the 1 g buffer during a collision of a 90 t wagon equipped with a 1 g buffer at a velocity of 7.2 km/h with a 80 t wagon at rest equipped with standard UIC-buffers.

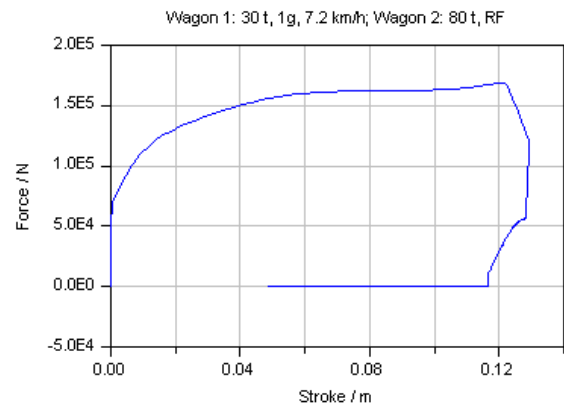


Figure 4: Force-stroke diagramm of the 1 g buffer during a collision of a 30 t wagon equipped with a 1 g buffer at a velocity of 7.2 km/h with a 80 t wagon at rest equipped with standard UIC-buffers.

multiple unit at a speed of up to 5 km/h with another one.

Furthermore it has to be proven that the coupler is pulled down correctly during a major impact with another S-Bahn up to a speed of 36 km/h and that the side buffer are capable of absorbing the remaining energy. The same proof has to be provided for a collision at 36 km/h against a freight car of mass 80 t. First the central coupler retracts and after that breaks away so that the laterally mounted auxiliary buffers take up the remaining energy. The entire process including pressure build-up, opening of the hydraulic predetermined breaking point, retraction of the damper, compression of the spring elements and deformation of the crash elements can be simulated in a single run.

The damper of the central coupler is a multifunc-

tional device (Figure 6). It contains a gas spring and a multiplier valve. They ensure that the coupler transmits the momentum and secondly absorbs enough energy to prevent the wagons from oscillating during the journey. During coupling at a speed of 5 km/h the damper has to absorb the total energy over a length of 140 mm without the force increasing above 1200 kN. If the force increases above 1500 kN, a hydraulic breaking point will be activated so that the coupler is retracted faster. In addition, the coupler comes with a return stroke damping, which prevents breakaway during run-up.

Every one of these scenarios has been simulated with a dynamical model for both multiple units. Thereby the flexibility of the car body, its connection to the bogie and the behaviour of the short couplers between cars have to be modelled with sufficient precision

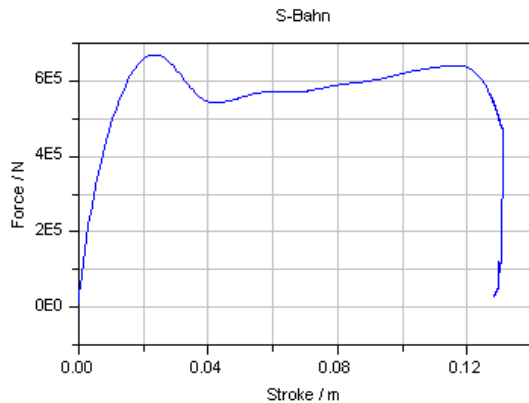


Figure 5: Force-stroke diagram for an entire S-Bahn multiple unit consisting of 6 double decker coaches with a total mass of 312 t and a total length of 150 m

4 Future Work

A follow up project (called DyMorail2) has been funded by CTI and will be carried out in collaboration with Schwab Verkehrstechnik AG. We intend to implement the following improvements to the first DyMoRail library:

1. Each buffer and coupler should be modelled in different levels of detail and complexity, in order to gain flexibility for simulating entire compositions consisting of several cars on the one hand and single wagons on the other.
2. The valve control has to be redesigned. At present the valve is modelled such that it opens

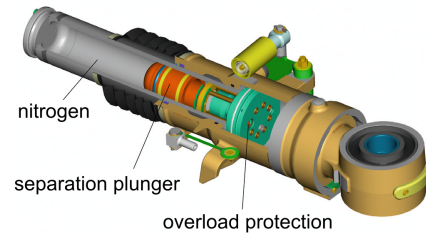


Figure 6: Construction drawing of the damper showing its working principle. It shows the damper bush in brass colour. On the rear is the air spring filled with nitrogen. The front part contains oil and both chambers are separated by a movable piston ("separation plunger" in red). In green is shown the overload pressure valve.

at a certain pressure and closes again at a lower one. This model is very simple and robust and can be used for a broad spectrum of applications. However, for modelling of long trains equipped with hydraulic buffers in combination with spring buffers these valves produce a lot of events, which increases the simulation complexity.

3. We also plan to implement crash scenarios according to new European norm DIN EN 15227. It contains requirements to the construction of rolling stock in order to minimize the consequence of collisions. It applies to the car body as well as to coupler and buffer.

5 Conclusion

With DyMoRail1 a powerful library has been implemented which allows to simulate longitudinal dynamics of entire railway trains. With this library an efficient simulation of complete train compositions in various combinations is possible. Modelica/Dymola has the following advantages over other tools such as SIMULINK or STELLA:

- Every model can be reused immediately
- Cars, buffers, crash elements and couplers can be arbitrarily combined
- Little effort is needed to establish, test and reconfigure new models
- Documentation and filing of simulation experiments is straight forward

- Even non-experts can carry out simulations with DyMoRail

This simulation environment was successfully used by Schwab Verkehrstechnik AG during the development of their state-of the-art center coupler product family. Within DyMoRail2 we intend to implement further features and improve the modularity and flexibility of the library.

References

- [1] Maurer W. Puffer nach Mass. Eisenbahn Revue 3, 2003, p.118-119.
- [2] Maurer W. Simulationsgestützte Entwicklung von Puffern und Dämpfern für Eisenbahnzüge. Proceedings of the 18th Symposium on Simulationstechnique ASIM 2005, Erlangen, Germany, ASIM September 12-15 2005.

Natural frequency analysis of Modelica powertrain models

Garron Fish

Mike Dempsey

Juan Gabriel Delgado

Neil Roberts

Claytex Services Ltd

Leamington Spa, United Kingdom

garron.fish, mike.dempsey, juan.delgado, neil.roberts @claytex.com

Abstract

The natural frequency analysis of complex powertrain models created in Modelica presents a number of problems. This paper presents the basic principles and some of the problems associated with carrying out this kind of analysis. As a result of this work, a new feature in the Powertrain Dynamics Library has been developed to automate these methods and provide the end-user with a simple set of functions to perform natural frequency analysis. Simple examples are used to illustrate the problems and solutions and a complex powertrain model is then analysed using the library.

Keywords: modal analysis; natural frequency; linearization; powertrain; NVH

1 Introduction

Modal analysis is the study of the dynamic response of a system at its resonance frequencies. Modal analysis is used in many fields for example in structural engineering to design buildings resistant to earthquakes [1] and in vehicle powertrain design to avoid poor NVH characteristics [2].

For a vehicle, modal analysis is carried out on all parts of the car to determine their natural frequencies. Care is taken to make sure that the natural frequencies of the parts in the car are all at distinct, separate frequencies. If the natural frequencies are not suitably separated this can lead to resonance across multiple parts of the car and a poor NVH characteristic.

A new feature has been introduced in the PTDynamics library [3] [4] to perform the natural frequency analysis of powertrain models created using this library. This paper highlights some of the problems involved with this type of analysis based on Modelica models and discusses some of the techniques developed to solve these.

To determine the natural frequencies of a model and the corresponding modal response we start by

linearising the model at the required operating point. Linearisation of a model using Dymola returns the state-space representation of the model and from this the natural frequencies can be calculated. The natural frequencies are found when all damping in a model is removed.

2 Modal frequency analysis and Modelica models

2.1 Basic Principles

This section looks at the basic modal analysis principles applied to a spring mass network. The example of a spring mass network has been chosen so that the natural frequency of a model can be described. An unforced spring mass network can be represented by the following ordinary linear differential equation:

$$M\ddot{s} + C\dot{s} + Ks = 0$$

It is common to calculate the natural frequency of the above equation with the damping term set to zero so the equation becomes:

$$M\ddot{s} + Ks = 0 \quad (1)$$

The natural frequency of the spring mass system can now be calculated from the roots of the above equation. The roots are the eigenvalues and eigenvectors of the equation.

To perform modal analysis on complex models we linearise these first which generates the state space representation of the model. The state space representation of a model is given by:

$$\begin{aligned} \dot{x} &= Ax + Bu \\ y &= Cx + Du \end{aligned} \quad (2)$$

where:

A, **B**, **C** and **D** are matrices

u is the vector of inputs

\mathbf{y} is the vector of outputs
 \mathbf{x} is the vector of states

To rearrange our simple spring-mass system in to state space form is done by transforming equation (1) in to the following form:

$$\ddot{\mathbf{s}} = -\mathbf{M}^{-1}\mathbf{K}\mathbf{s}$$

In this simple example, there are no inputs so the \mathbf{u} term is dropped and there are no outputs so the equation for \mathbf{y} is not required. The model is then reduced to:

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} \tag{3}$$

where

$$\mathbf{x} = \begin{bmatrix} \mathbf{s} \\ \dot{\mathbf{s}} \end{bmatrix}$$

and

$$\mathbf{A} = \begin{bmatrix} 0 & 1 \\ -\mathbf{M}^{-1}\mathbf{K} & 0 \end{bmatrix}$$

2.2 Eigenvalues and eigenvectors

For a given matrix \mathbf{A} the eigenvalues and eigenvectors are calculated such that:

$$\mathbf{A}\mathbf{v} = \lambda\mathbf{v}$$

where:

\mathbf{v} is the eigenvector associated with the eigenvalue
 λ is an eigenvalue

The eigenvalue solutions, are the roots of:

$$[\lambda\mathbf{I} - \mathbf{A}]\mathbf{x} = \mathbf{0}$$

All the eigenvalues are included in vector $\boldsymbol{\lambda}$ that is referred to as the eigenvalues of \mathbf{A} . The eigenvectors are combined row wise into matrix \mathbf{v} . The eigenvectors and eigenvalues of this equation are calculated so that the natural frequency can be calculated as follows in section 2.3.

2.3 Frequency and damping

The natural frequency is calculated from the eigenvalues as [5]:

$$f = \frac{|\boldsymbol{\lambda}|}{2\pi}$$

where:

$\|\cdot\|$ is the complex norm

f is frequency in Hertz

The complex norm is the sum of the squares of the real and imaginary parts all square rooted. There is also a damping term that is associated with each eigenvalue. In the case where the damping has been set to zero, this term will be zero and will not influence the natural frequencies of the model. The damping term can be calculated with the following equation [6]:

$$\zeta = \begin{cases} 0, |\lambda| = 0 \\ \frac{\text{Re}(\lambda)}{|\lambda|}, |\lambda| \neq 0 \end{cases}$$

where:

$\text{Re}(\cdot)$ is the real part of a complex number

The frequency that a model with damping oscillates at without being driven by an outside force is referred to as the *damped frequency* and using eigenvalue analysis this is calculated as :

$$w_d = w_n\sqrt{1 - \zeta^2}$$

2.4 Issues for complex Modelica models

The current analysis described above can be easily performed on a spring mass network but it is not as easy to implement this on a complex Modelica model. A number of issues arise when trying to apply this process using a Modelica tool such as Dymola.

A complex model will contain a large number of state variables and we would normally expect to find many states that do not have any effect on the natural frequency response of the physical states of the model. For example, states within a driver model or control system that do not directly influence the physical response of the system. These states should be removed from the analysis to reduce the time taken to do the analysis.

Some Modelica tools are able to compile models using dynamic state selection. Currently models that use dynamic states cannot be analysed and a fixed set of states needs to be applied to the model. This has to be done by the user before starting the frequency analysis.

In the simple spring mass network presented so far we have not considered the possibility of the relative state of the spring being selected as a state rather than the position of the mass. Modelica tools are able to select a set of states from a model and in many cases they will select relative states rather than absolute states. Whilst the natural frequencies of the

system are unaffected by the choice of state variable it is preferable in this type of analysis to use the absolute states of the system. Using the absolute states makes the interpretation of the modal response easier as the points of interest become physical points such as the driveshaft ends or pinion gear rather than relative states such as the driveshaft twist or relative angle between pinion and crown wheels.

Further problems are observed when component models that utilize the standard Modelica friction model are included for analysis. The behaviour of the slip/stick friction models is not linearized in the expected manner and modifications to the analysis have to be made around these components.

To calculate the natural frequencies the damping terms have to be removed from the model but without the damping often models will not simulate. This causes a problem for the initialisation of models and when the model needs to be analysed under different operating conditions, for example, in different gears or under different loading conditions where springs are compressed to different parts of their non-linear force curve.

3 Implementation in the Powertrain Dynamics Library

The Powertrain Dynamics (PTDynamics) library is used to create complex MultiBody models of powertrains in a user friendly and efficient manner. A new feature has been introduced to determine the natural frequencies of these powertrain models. A number of issues are present that make performing the natural frequency analysis difficult when working with Modelica models (refer to Section 2.1). This section describes some of the methods implemented in to the linearization functions available in the PTDynamics library that are used to overcome these issues.

3.1 Relative states

The natural frequencies of the model are typically calculated for positional states (i.e. position or angular position). However when a model is created using Modelica, the modelling tool can choose to select relative states (such as spring extension) rather than positional states (such as the position of ends of the spring). When this is detected in a model the relative states are converted in to positional states before linearizing the model.

The first step in the analysis process is to determine the states used in the model which is done by translating the model and analysing the list of

selected states. If relative states are detected then the model has to be modified by adding outputs that measure the positions either side of the component with the relative states, see Figure 1. The model can then be linearized and the resulting A matrix manipulated to transform the relative state in to a positional state. Within the PTDynamics library a precise naming convention is used to enable the automatic detection of relative and absolute states from the variable names.

By only making the transformation from relative to positional state in the linearized model we do not affect how the original model simulates. This means that we can still use the original model to get the system to the desired operating point and then linearize it. If we forced the user to only use

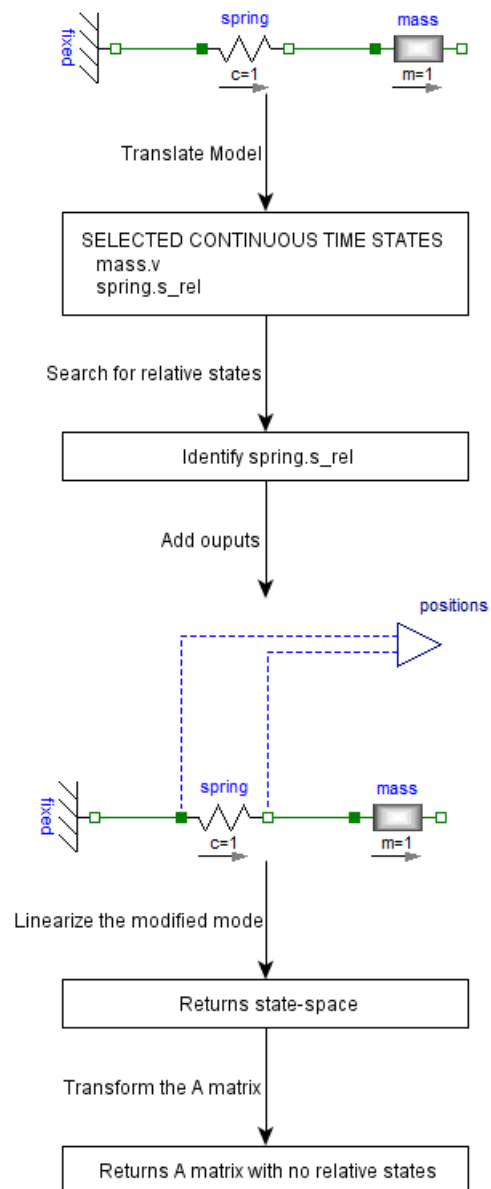


Figure 1: Process to convert relative states to positional states

positional states in the model we may introduce slight differences in to the model due to the different equation solutions required and we could impact the simulation time.

When the modified model, with the added outputs, is linearized, the resulting state space representation includes these outputs in the **C** matrix. This matrix relates the position outputs to the states in the model. Each relative state will generate two outputs but only one of these outputs will be related to the relative state by the **C** matrix. This state is used to replace the relative state.

Using the spring mass model as an example we can see how this manipulation of the **A** matrix should be performed. Linearizing the modified model gives the following:

$$A = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$$

$$C = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$$

state names = {mass.v, spring.s_rel}
output names = {position1, position2}

From the **C** matrix it is seen that **position2** is related to **spring.s_rel** as:

$$\mathbf{position2} = C[2, :] \mathbf{x}$$

where:

x are the states of the model

A transformation matrix is now created that transforms **x** to a set of states that does not contain relative states. In this example the transformation matrix would be:

$$T = \begin{bmatrix} 1 & 0 \\ C[2,1] & C[2,2] \end{bmatrix}$$

$$\mathbf{x}_{pos} = T \mathbf{x} \tag{4}$$

Replacing **x** in (3) with **x_{pos}** from (4) gives:

$$\dot{\mathbf{x}}_{pos} = T A T^{-1} \mathbf{x}_{pos}$$

A drawback of this method is that it can select a state that is only associated with a position and not directly with an actual mass or inertia state. Figure 2 illustrates a case where this behaviour is present. The user currently has to review the selections made

during the analysis process to ensure that these situations are avoided.

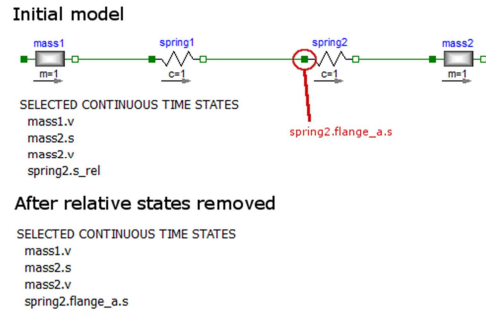


Figure 2: The initial states of the model include **spring2.s_rel**, this state is replaced with **spring2.flange_a.s** that is a state without a mass

3.2 Friction components

A number of component models such as clutches and brakes use the Modelica Standard Library coulomb friction model [7] that handles the stuck and sliding modes in a clean way using state events. When this is linearized using the built-in Dymola function the model is sometimes linearized as if in the slipping mode regardless of the actual state of the component. A method has been developed to adjust the model and resulting state space model to correctly account for the friction state.

Figure 3 shows an overview of the automatic process that is used to overcome this using the PTDynamics library. First the model is translated and the names of the selected states are analysed to determine if there are any states that relate to friction and to determine what state the friction model is in at the instant that the model is being linearized at.

If the friction model is in the stuck mode then it is necessary to join the positional states in the **A** matrix that are either side of the frictional component. To be able to join states in the **A** matrix it is necessary to calculate the mass/inertia of the states being joined together. This is done by adding torque inputs to the corresponding positional states either side of the friction component.

In the example shown in Figure 3, we would detect the friction states within the clutch and then modify the model. In addition to adding a torque input either side of the friction model we also need to add position outputs either side of the friction model so that we can join the states in the locked mode.

After the modified model is linearized the \mathbf{B} matrix is used to determine the mass of the states. This information together with the state space \mathbf{C} matrix can then be used to update the \mathbf{A} matrix by joining the states on either side of the friction component.

The mass of the states is determined as follows, the basic equation describing a spring mass system that contains a force is:

$$m\mathbf{a} = k\mathbf{s} + d\mathbf{v} + \mathbf{F}$$

where:

m is mass

s is position

v is velocity

a is acceleration

k is stiffness

d is damping vector

F is the applied force

In the example shown in Figure 3, the positional states that the clutch is connected to are independent

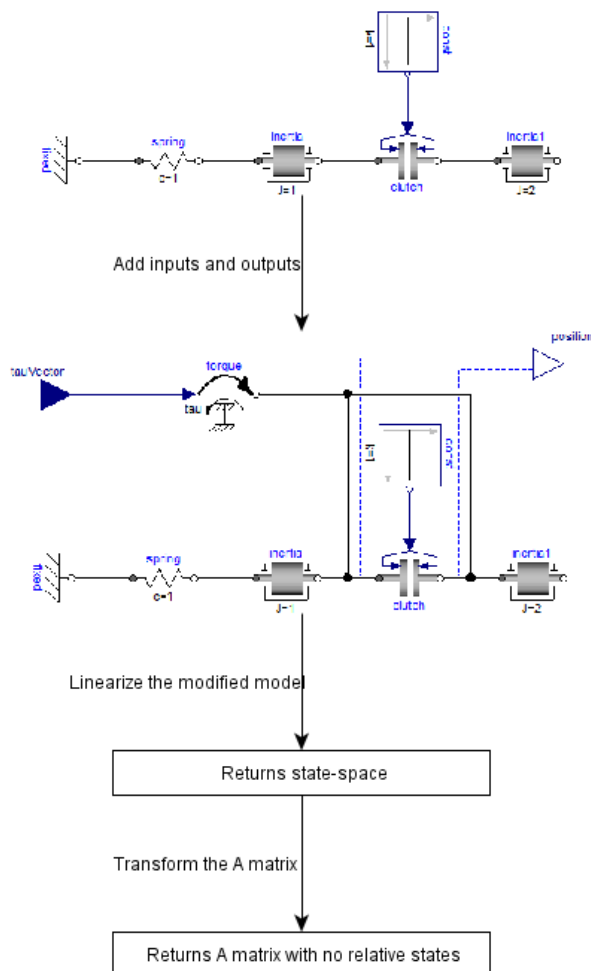


Figure 3: Process to handle friction components

which means the following equation can be used to describe both states that need to be joined together and rearranged as:

$$\ddot{s} = m^{-1}k\mathbf{s} + m^{-1}d\dot{s} + m^{-1}F$$

The state space representation of this equation is:

$$\begin{bmatrix} \dot{s} \\ \ddot{s} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ m^{-1}k & m^{-1}d \end{bmatrix} \begin{bmatrix} s \\ \dot{s} \end{bmatrix} + \begin{bmatrix} 0 \\ m^{-1} \end{bmatrix} F \quad (5)$$

From (2) and (5), we can determine that the state space \mathbf{B} matrix is equal to $\begin{bmatrix} 0 \\ m^{-1} \end{bmatrix}$, so the mass/inertia for the states to be joined can be calculated. Using the example shown in Figure 3, we get the following values when linearising the modified model.

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 0.5 \end{bmatrix}$$

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

state names = {i1.phi, i1.w, i2.phi, i2.w}

input names = {tau1, tau2}

output names = {position1, position2}

Using the \mathbf{B} matrix we can determine the inertia of the two bodies either side of the clutch.

$$m_1 = \frac{1}{B_{2,1}} = 1$$

$$m_2 = \frac{1}{B_{4,2}} = 2$$

To modify the \mathbf{A} matrix we use the \mathbf{B} matrix to determine the rows in the \mathbf{A} matrix that should be combined. The \mathbf{C} matrix is then used to determine the columns that need to be combined. After combining the rows and columns we can remove the redundant rows and columns from the \mathbf{A} matrix.

In this example we find that the 2nd and 4th rows need to be combined as well as the 1st and 3rd columns which results in:

$$A = \left[\begin{array}{cc} \frac{0+0}{(-1+0)*m_1+0*m_2} & \frac{1}{(0+0)*m_1+0*m_2} \\ \frac{1}{m_1+m_2} & \frac{1}{m_1+m_2} \end{array} \right]$$

$$A = \begin{bmatrix} 0 & 1 \\ 0.333 & 0 \end{bmatrix}$$

To include damping effects when joining states using this method the columns corresponding to the rows determined from the B matrix need to be added together as well.

There is a known limitation of the joining method demonstrated here and used in the PTDynamics library in that the states being joined together must be independent states. This means that the positional state must not be dependent on other positional states. An example of a component that has dependent states is a planetary gear where the rotational states of the three shafts are dependent on each other. To overcome this limitation a flexible shaft has to be connected between a clutch and a planetary gear in a gearbox to be able to join the states on either side of the clutch using this method.

4 Applications

4.1 Simple example

This simple example contains three inertias with the first two separated by a clutch and the second and third inertia separated by a spring as shown in Figure 4. A ramp input is used to actuate the clutch and goes from 0 at 0s to 1 at 1s. The response for the clutch state and the speeds of the inertias either side are shown in Figure 5.

If the model is linearized at $t=0s$, i.e. when the clutch is open we find the natural frequency is at 5.29Hz. If the model is linearized at $t=2s$, when the clutch is locked, the natural frequency occurs at 2.20Hz.

The change in frequency occurs because the total effective inertia on the left hand side of the spring has changed. Without using the method to join the states either side of the clutch the built in functions report no change in the natural frequency despite the change in configuration of the model.

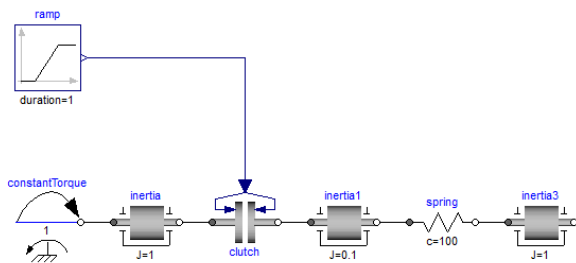


Figure 4. Simple model that contains a clutch and a spring

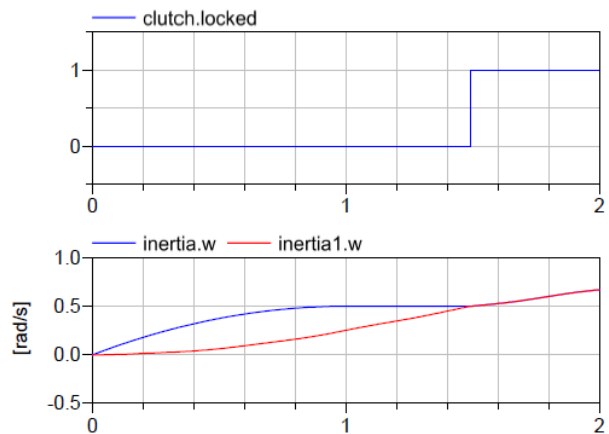


Figure 5. Plots of locked and angular velocity of inertia and inertia1 in the Simple model in Figure 4.

4.2 Full vehicle example

A model of a front engine, rear-wheel drive vehicle with a manual transmission was constructed using the PTDynamics library it fully test the new functions and methods. The model is shown in Figure 6. The engine model is a simple mapped engine model but the transmission and driveline are more detailed. Figure 7 shows the gearset model from within the transmission. The gearset and driveline models include torsional compliance in a number of the shafts but are rigidly mounted within the chassis. Overall this model has a good torsional representation of the powertrain system and would be suitable for studying driveability events such as tip-in and tip-out.

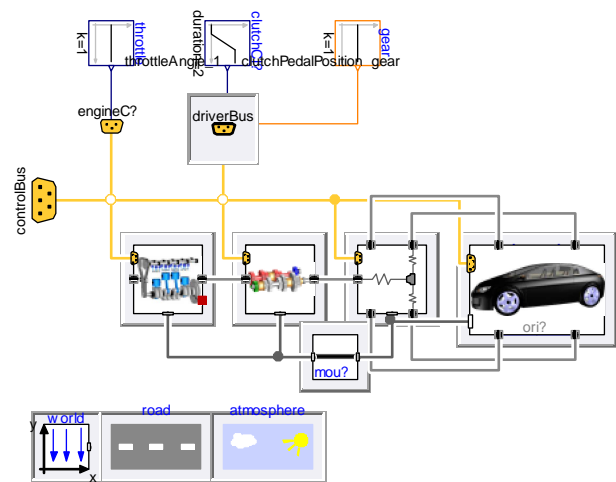


Figure 6. PTDynamics vehicle example that is linearized

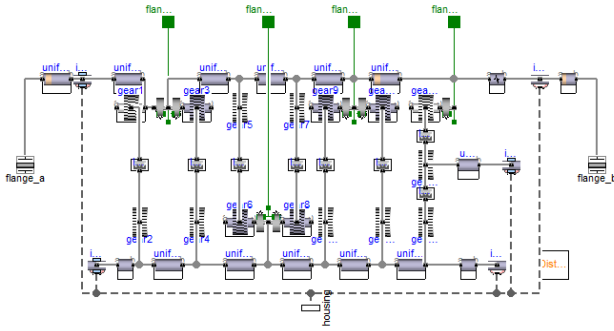


Figure 7. Gearset used in vehicle example.

The chassis model doesn't include suspension but the tyres do include a slip model based on the well-known Pacejka tyre model. This required the development of a method that relates the wheel rotation to the chassis movement. This was necessary because the slip models are based on velocity relationships but for this type of analysis we need the relationships to be based on position. The method developed assumes that the ratio between the wheel rotation and the chassis motion is a fixed ratio at the instance that linearization occurs. The details of this method are not described in this paper.

The model was linearized and the following natural frequencies are found (in Hz): 5.1, 35, 124, 266 and 343. The 5.1Hz response is the shuffle frequency of the vehicle and the modal response is shown in Figure 8. The x-axis of the modal response plots is an integer that corresponds to the states listed in Table 1. The magnitudes are normalised with respect to the variable with the largest displacement.

The modal response shows that at this frequency there is very little motion of the chassis but the whole powertrain is moving out of phase with the chassis and at relatively large displacements.

No.	State
1	transmission.clutch.drivenPlate.flange_a.flange.phi
2	transmission.gearset.uniformShaft10.body_a.phi
3	transmission.gearset.uniformShaft.body_a.phi
4	driveline.rearDifferential.pinion.phi
5	driveline.rearDifferential.differentialAssembly.outputGear_2.phi
6	chassis.motion.prismatic_x.s

Table 1. States of simple vehicle. Each number corresponds to a state. The number in the legends in Figure 8 corresponds to the number in this table.

It is also possible to generate Bode diagrams for different inputs and outputs of the vehicle model. The example shown in Figure 8 is the bode diagram generated when engine torque is an input to the system and the differential pinion gear rotation angle is the output. The Bode plotting function in

Modelica_LinearSystems2 is used to generate the actual plot.

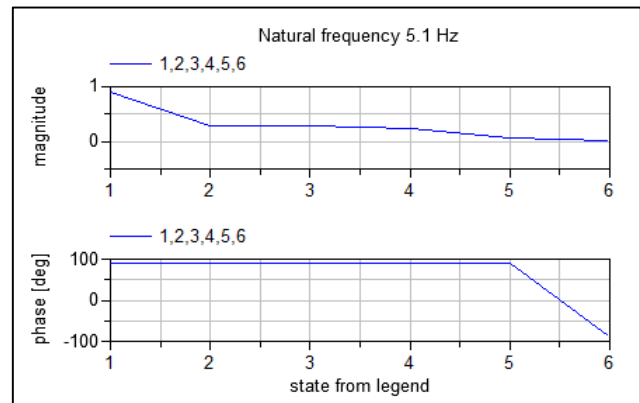


Figure 8. Modal response of the vehicle model at 5.1Hz. The magnitude and phase of the different states are plotted. Each state is assigned to a position along the x axis as determined by the legend. The numbers in the legends correspond to the states in Table .

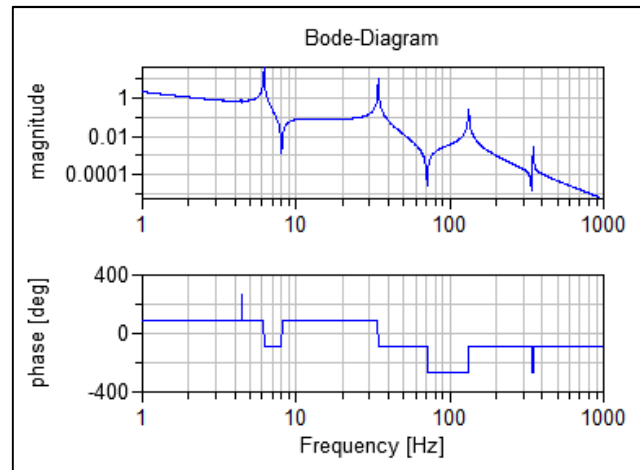


Figure 9. Bode diagram with Engine torque as the input and differential pinion position as the output.

5 Conclusion

A new method for determining the natural frequencies and modal responses of complex Modelica models has been developed and introduced as a new feature in the Powertrain Dynamics library. This feature includes automated methods to handle the problems with relative states and friction components as described in this paper in addition to other methods to handle further problem areas such as tyre slip models. The feature will be further improved to provide animation of the modal response of the powertrain to aid the understanding of the natural frequencies of the powertrain system.

References

- [1] M. Paz, "International handbook of earthquake engineering," in *International handbook of earthquake engineering*, London, Chapman & Hall, 1994, p. 283.
- [2] H. Gagnon and G. Gagnon, "Identifictation of powertrain noise sources using sound intensity and modal analysis techniques," *Proceeding of SPIE, the International Society for Optical Engineering*, vol. 3089, no. 2, pp. 2016-2020, 1997.
- [3] N. Roberts and M. Dempsey, "Predicting the launch feel of automatic and dual clutch transmissions," in *Proceedings 9th Modelica Conference 2012*, Munich, 2012.
- [4] M. Dempsey and A. Picarelli, "Investigating the MultiBody dynamics of the complete powertrain system," in *Proceedings 7th Modelica Conference*, Como, 2009.
- [5] MSC Software, "MD Nastran 2010, Dynamic Analysis User's Gude," 2010, p. 47.
- [6] A. Mohamed, "Modelling, simulation and identification," Rijeka, Sciyo, 2010, p. 68.
- [7] M. Otter, H. Elmqvist and S. E. Mattsson, "Hybrid Modeling in Modelica based on Synchronous Data Flow Principle," in *CACSD'99*, Hawaii, 1999.

Achieving $O(n)$ Complexity for Models from Modelica.Mechanics.MultiBody

Christian Schubert^a Jens Frenkel^a Günter Kunze^a Michael Beitelschmidt^b

^a Professur für Baumaschinen- und Fördertechnik

^b Professur für Dynamik und Mechanismentechnik

Technische Universität Dresden

01069 Dresden, Germany

{christian.schubert, jens.frenkel, guenter.kunze

michael.beitelschmidt}@tu-dresden.de

Abstract

This paper presents a graph theoretical interpretation of the well-known $O(n)$ algorithm for Multibody systems. It enables Modelica compilers to solve for the unknown accelerations of a Multibody model without the need of inverting a dense mass matrix which would require $O(n^3)$ operations.

Keywords: MultiBody, Relaxation, Gaussian Elimination, OpenModelica

1 Introduction

Simulation has become an indispensable tool in early development stages. Increasing computational power leads to a demand for more detailed models. Especially in the design of Mobile Machinery, Multibody systems are of major importance.

Currently, most Modelica compilers apply Tearing [1] to models from Modelica.Mechanics.MultiBody yielding a dense linear system of size proportional to n - the number of bodies. In order to solve for the unknown joint accelerations the system has to be inverted which requires $O(n^3)$ operations. Hence this approach is only recommendable for small to medium sized problems.

Efficient algorithms with $O(n)$ complexity are well known from literature [2], [4]. Unfortunately their application for Modelica.Mechanics.MultiBody proves to be difficult since these algorithms rely on special knowledge about the multibody systems which is not available in a general equation based framework like Modelica.

It has already been pointed out in the literature [5] that a technique called *Relaxation* is able to yield such

an $O(n)$ formalism for multibody systems. However, adaptations to the model libraries as well as a specific model structure were required.

This paper presents a novel algorithm for general purpose Modelica compilers. It is based on a graph theoretical generalization of the well known $O(n)$ algorithm for multibody systems adapted to models from Modelica.Mechanics.MultiBody.

2 Multibody systems

2.1 Kinematic Graph

Every multibody system can be represented by a kinematic graph whose nodes represent both bodies and inertial frames and whose edges correspond to joints. If the kinematic graph contains closed loops, appropriate joints, so called *cut-joints*, are temporarily removed so that the resulting graph only consists of *trees*. In a tree, every node (body) has a unique parent, which is the next node on the path to the root (inertial frame). All bodies are numbered, such that every child body has a higher number than its parent. Each joint is numbered according to the child body it is connected to, thus forming pairs of bodies and joints. All remaining cut-joints are numbered consecutively. An example is given in Figure 1.

2.2 Equations of Motion

The equation of motion of a single body i can be written as

$$\mathbf{M}_i \mathbf{a}_i = \mathbf{p}_i + \mathbf{f}_i - \sum_{k \in \mu(i)} \mathbf{R}_{k,i}^T \mathbf{f}_k \quad (1)$$

$$\mathbf{p}_i = \mathbf{f}_{i,ext} - \mathbf{h}_i \quad (2)$$

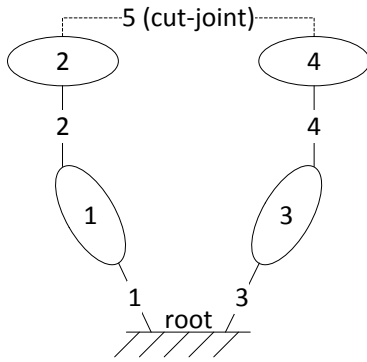


Figure 1: Kinematic Graph of Example System

where $\mathbf{M}_i \in \mathbb{R}^{6 \times 6}$ represents the mass matrix, $\mathbf{a}_i \in \mathbb{R}^6$ both translational and rotational acceleration of a fixed point on body i and $\mathbf{h}_i \in \mathbb{R}^6$ all gyroscopic terms. \mathbf{p}_i is used as an abbreviation for \mathbf{h}_i and all external forces and torques $\mathbf{f}_{i,ext}$. $\mathbf{f}_i, \mathbf{f}_k \in \mathbb{R}^6$ represent the joint reaction forces of the joint belonging to the body i as well as all the set of all its children $\mu(i)$. $\mathbf{R}_{k,i}$ transforms the force and torque from i to k whereas its transpose performs the opposite transformation.

It is assumed that every joint i has a set of joint-coordinates \mathbf{q}_i as well as joint velocities \mathbf{s}_i which fully determine its kinematic state. Thus, the acceleration of body i is given by

$$\mathbf{a}_i = \mathbf{R}_{i,h} \mathbf{a}_h + \mathbf{J}_i \dot{\mathbf{s}}_i + \mathbf{c}_i \quad (3)$$

where h is the index of the parent body of i . \mathbf{J}_i describes the degrees of freedom of joint i and \mathbf{c}_i collects all remaining terms which are neither linear in \mathbf{a}_h nor $\dot{\mathbf{s}}_i$.

From d'Alamberts Principle it can be found that

$$\mathbf{J}_i^T \mathbf{f}_i = \boldsymbol{\tau}_i \quad (4)$$

with $\boldsymbol{\tau}_i$ being the motor force driving the joint.

Since equations (1)-(4) are linear with respect to the accelerations and forces, one can merge the equations for every element of the multibody system into one single linear system of equations.

One of the most efficient $O(n)$ algorithms (see [3]) to solve this linear system of equations is defined through repeated application of

$$\dot{\mathbf{s}}_i = \boldsymbol{\rho}_i^{-1} (\boldsymbol{\tau}_i - \mathbf{J}_i^T \mathbf{M}_i^A (\mathbf{a}_{\lambda(i)} + \mathbf{c}_i) - \mathbf{J}_i^T \mathbf{p}_i^A) \quad (5)$$

$$\mathbf{a}_i = \mathbf{a}_{\lambda(i)} + \mathbf{J}_i \dot{\mathbf{s}}_i + \mathbf{c}_i \quad (6)$$

requiring the calculation of the following variables for each body starting at the highest index

$$\mathbf{M}_i^A = \mathbf{M}_i + \sum_{k \in \mu(i)} \mathbf{M}_k^A \quad (7)$$

$$\boldsymbol{\rho}_i = \mathbf{J}_i^T \mathbf{M}_i^A \mathbf{J}_i \quad (8)$$

$$\mathbf{M}_i^a = \mathbf{M}_i^A - \mathbf{M}_i^A \mathbf{J}_i \boldsymbol{\rho}_i^{-1} \mathbf{J}_i^T \mathbf{M}_i^A \quad (9)$$

$$\mathbf{p}_i^A = \mathbf{p}_i + \sum_{k \in \mu(i)} \mathbf{p}_k^a \quad (10)$$

$$\mathbf{p}_i^a = \mathbf{p}_i^A + \mathbf{M}_i^A \mathbf{c}_i + \mathbf{M}_i^A \mathbf{J}_i \boldsymbol{\rho}_i^{-1} (\boldsymbol{\tau}_i - \mathbf{J}_i^T \mathbf{p}_i^A) \quad (11)$$

It can be shown that this exact algorithm can be derived from a (sparse) Gaussian Elimination of the linear system of equations provided all equations and variables are ordered correctly.

In a general equation based framework, information such as the ordering of bodies is not readily available. Thus the algorithm cannot be applied directly. However, [5] has shown that the application of a technique called *Relaxation*, which is a type of Gaussian Elimination, may also lead to an $O(n)$ algorithm. The suitable ordering of the equations and variables was achieved by inserting a special *relax* operator into the model equations.

This paper follows another path in which the $O(n)$ algorithm is derived using graph theoretical techniques. To do so a graph representing the equations of motion is built from the multibody system. The key to an efficient $O(n)$ algorithm lies in the ordering of the graph. This is a problem for a Modelica compiler since the information about the structure is lost in the compilation process but is needed to achieve the efficiency of algorithms such as [3]. By trying to generalize the idea behind the algorithm from [3], a good ordering for general modelica models can be found which consequently leads to an efficient $O(n)$ algorithm. This approach is described in the following section.

3 Graphtheoretical Interpretation

3.1 Graph of a system of equations

Given a set of equations, an undirected bipartite graph can be defined which contains two sets of nodes representing equations and variables respectively. There is an edge between a variable and an equation if and only if that equation depends on that variable. The graph of the equation system belonging to the example system has been sketched in Figure 2. Every square node represents an equation whereas every circle represents a variable. Nodes representing eq. (1) have been named I_i , eq. (3) is called A_i and (4) is labelled D_i . In addition to the definition above, the edges carry the partial derivative of the equation with respect to the variable.

It can be seen that the graph exhibits two *legs*. The first leg contains all kinematic quantities (A_i, a_i)

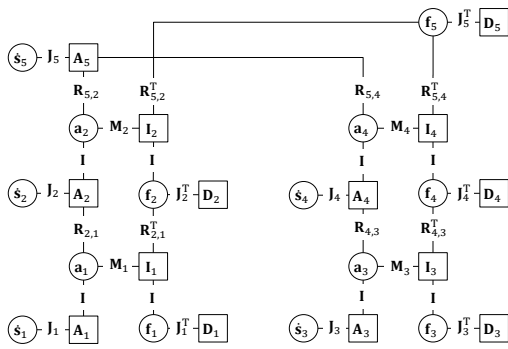


Figure 2: Equation Graph of Example System

whereas the second leg comprises all kinetic quantities (I_i, f_i). The two legs are interconnected through *steps* given by the inertial equations I_i (eq. 1). Equations D_i (eq. 4) and variables \dot{s}_i appear as *handles* to the legs, thus forming a ladder like structure. All nodes with the same index represent a body along with its joint and shall be denoted as *body structure*. A body structure is called *terminal* if the body it represents does not have any children.

3.2 Gaussian Elimination

Gaussian Elimination can be applied to a linear system of equations $\mathbf{Ax} = \mathbf{b}$. Therefore one has to reproduce \mathbf{A} from the equation graph of the multibody system. This requires the numbering of all equation and variable nodes, i.e. allocating them to rows and columns of \mathbf{A} . The algorithm then iterates over all elements on the main diagonal of \mathbf{A} , which are called *pivots*. Multiples of the current row are added to all rows below such that all elements below the pivot are eliminated. Thus \mathbf{A} is reduced to $\hat{\mathbf{A}}$ which has a (block) upper triangular form.

Given the numbered graph, Gaussian Elimination can be applied directly:

1. Begin at $i = 1$
2. Check that there is an (invertible) edge between equation node i and variable node i (equivalent to pivot element)
3. Create Edges between all pairs of equations and variables connected to equation and variable nodes i
4. Remove equation and variable nodes i along with all adjacent edges from the graph

5. Continue at 2 with $i := i + 1$ until all nodes are removed

Figure 3 shows this process for a body structure (see section 3.1) as found in Figure 2.

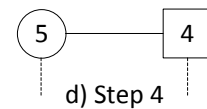
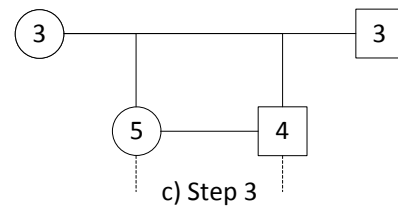
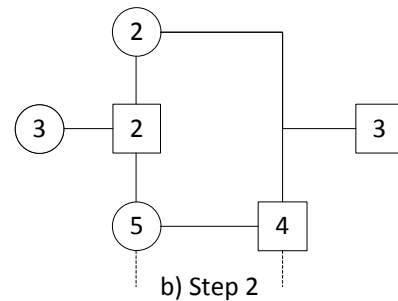
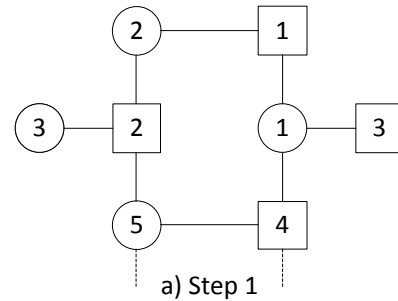


Figure 3: Steps of Gaussian Elimination

3.3 $O(n)$ algorithm

The numbering shown in Figure 3 leads to the efficient $O(n)$ algorithm from [3].

Removing the closed loop (A_5, D_5, f_5, \dot{s}_5) from the graph given in Figure 2 yields two terminal body structures. These can be eliminated as shown in Figure 3 revealing new terminal body structures. This process can be repeated until all body structures are eliminated.

Looking at the numbering employed in Figure 3 one may note that

1. Resulting pivots are chosen to be identity matrices if possible
2. Each body structure is treated separately, beginning at the terminal ones
3. All nodes between the handles of the body structure are being numbered consecutively beginning at the equation handle
4. Equation and variable handle are given the same number although there is initially no connection between them (zero pivot)
5. Entries in the lower triangular part of \mathbf{A} only occur due to the D_i nodes as well as the I_i nodes of non terminal body structures.

Please note that the handles nodes correspond to a suitable choice of tearing variables and residual equations, as described in [1].

One may expect that the application of these rules to the equation graph found in models from Modelica.Mechanics.MultiBody may yield a numbering which leads to an efficient $O(n)$ algorithm for multi-body systems for a general purpose Modelica compiler.

4 Application to Modelica.Mechanics.MultiBody

4.1 Equation Graph

Due to the object oriented nature of Modelica.Mechanics.MultiBody the equations are the same as in 2.2 but are not written in such a compact form. Equation (1) is found in the *Body* model. Equations (3) and (4) are found in the different joint models. The transformation matrices $\mathbf{R}_{i,k}$ (see Eq. (1)) are defined through the *FixedTranslation* and *FixedRotation* models. The linear system of equations under consideration is found as a strong connected component through Tarjan's algorithm [8] after index reduction has been applied [7]. Moreover, most Modelica compilers apply symbolic simplifications to the equations of motion. Figure 4 shows the graph of the sample system with which a Modelica compiler has to deal with.

Application of the $O(n)$ algorithm requires three steps:

1. Recover the graph structure
2. Find a suitable ordering

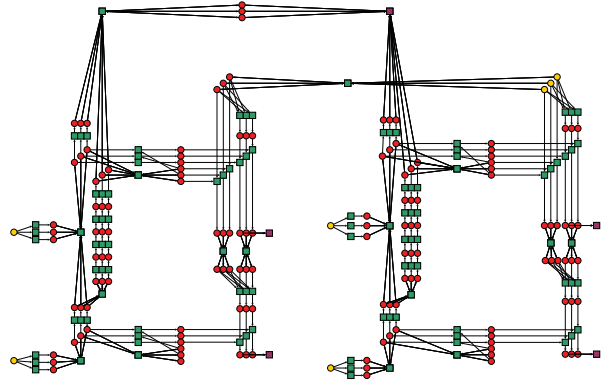


Figure 4: Equation Graph of Example System modelled with Modelica.Mechanics.MultiBody

3. Apply Gaussian Elimination

Every step will be discussed in the following.

4.2 Tree Structured Systems

4.2.1 Recovering the graph structure

The first rule (see section 3.3) says, that if possible the pivots shall be chosen to be identity matrices. Therefore pairs of equations and variables have to be found whose partial derivative is an identity matrix. This process shall be called *Natural Matching*. In a first step, every vectorial equation is tested if it can be solved for its unmatched vectorial variables with only using addition and subtraction. If that is the case, this equation and variable are *matched*. Afterwards, all remaining equations and variables are expanded to their scalar representation. All remaining scalar equations are tested if they can be solved for their unmatched vectorial variables with only using addition and subtraction. If that is the case, this equation and variable are also *matched*. Then a classic matching algorithm [6] is applied, leaving a set of variables and equations unmatched. These are the candidates for the tearing variables and residual equations. This procedure has already been suggested in [9].

Since all equations in the Modelica.Mechanics.MultiBody library have been written down in a manner which is most suitable for computation, it happened in all our tests that the set of candidates may be used without further modification as tearing variables and residual equations. The tests also showed, that mostly joint accelerations were used as tearing variables and, depending on symbolical simplifications, the D_i equations or close neighbours were used as residual equations.

The result of the matching algorithm is visualized in the graph by assigning directions to all edges. An edge from an equation to a variable means that this equation is used. The result of the matching algorithm is visualized in the graph by assigning directions to all edges. An edge from an equation to a variable means that this equation is used to calculate that variable. An edge from a variable to an equation means that this variable is needed in the calculation of that variable. All tearing variables are assumed to be known whereas all residual equations do not have any variables that they are solved for. The result for the example system including the kinematic loop is shown in Figure 4.

Next, the order between the tearing variables has to be found. Therefore the algorithm starts at a tearing variable and follows the edges in opposite direction, thus running down the kinematic leg. When another tearing variable is found, it must be the predecessor and the traversal is stopped. Thus, the predecessor to every tearing variable can be found defining an order between them which corresponds to the kinematic graph of the mechanical system. Please note, that this only works for tree structured systems. Otherwise a body, and therefore a tearing variable, may have more than one predecessor.

In a next step, the residual equation to each tearing variable has to be found. Again, a breadth-first graph traversal is started from every tearing variable following each edge. The first residual equation, that is found is assigned to the tearing variable.

4.2.2 Finding a suitable ordering

From the kinematic graph, obtained in the previous step, the terminal pairs of tearing variables and residual equations are known. Starting at a terminal residual equation all paths to its tearing variable can be found by following the in opposite direction. Valid paths may also include eliminated nodes. Once all paths have been found, decreasing numbers are assigned to the nodes using a breadth-first-search starting at the tearing variable. Afterwards the nodes of the residual equation and the tearing variable are numbered. Then all numbered nodes are eliminated from the graph as well as the tearing variable from the kinematic graph. This process is repeated for the next terminal tearing variable until all tearing variables are eliminated. In a last step all remaining nodes are numbered. Thus, a number has been assigned to every node allowing to apply Gaussian Elimination.

4.2.3 Applying Gaussian Elimination

Given the numbering of all nodes, the matrix \mathbf{A} can be constructed. Next Symbolic Gaussian Elimination is applied to the matrix $\mathbf{G} = \begin{bmatrix} \mathbf{A} & \mathbf{b} \end{bmatrix}$ yielding an upper-triangular \mathbf{G}' , see section 3.2. The equations of the strong connected component are then replaced by $\mathbf{x} = \mathbf{G}'^{-1}\mathbf{b}$.

When performing Gaussian Elimination temporary variables should be introduced after every elimination step. Otherwise the symbolic expressions in the entries of \mathbf{G}' may grow very fast.

4.3 Closed Loop Systems

Adapting the algorithm to cope with closed kinematic loops is part of the ongoing work. This section shall outline the problem and possible solutions.

Natural Matching still works reasonably well, choosing joint accelerations and the constraint forces of the loop as tearing variables. The search for the predecessors of the tearing variables, however, breaks down. Firstly because the tearing variables of the kinematic loop may have more than one predecessor and secondly because they are located in the kinetic leg.

The ordering between the tearing variables then has to be modified such that, the tearing variables of the loop closure joint are treated after all other tearing variables belonging to the same kinematic loop.

So far, the search for the predecessors has been extended so that it finds every tearing variable which is a parent in the kinematic graph. It leads to a dramatic increase in effort for both the search as well as the assembly of the kinematic graph. Tests have shown that the whole algorithm succeeds for some models, including the sample model, but it fails for others. Failing is mainly caused because the search for predecessors sometimes returns unexpected results.

5 Numerical Tests

The described algorithm has been implemented into the OpenModelica Compiler. It has been tested on multibody systems with tree structure only.

The following models have been used for testing:

1. Planar Pendulum - A sequence of n submodels consisting of a revolute joint, a body and a fixed-Translation
2. Split Pendulum - Same as Planar Pendulum but with a short extra branch of constant length

3. Alternating Pendulum - Same as Planar Pendulum, but with alternating axes of rotation
4. Multi Pendulum - Each body is followed by two more pendulum bodies with a limited recursion depth (see Figure 5)

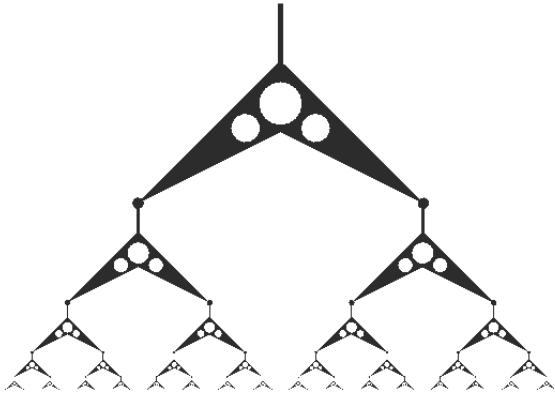


Figure 5: Multi Pendulum

Figures 6, 7, 8 and 9 show the required operation counts needed to calculate the whole model (including the accelerations) for the four test cases. As can be seen each curve exhibits a linear dependence on the number of bodies and therefore the number of degrees of freedom.

For comparison, the results when using tearing [1] which is ($O(n^3)$) have also been included. One may see that for planar systems the $O(n)$ algorithm produces much lower operation counts as the number of bodies grow. In the 3D case, however, the tearing algorithm outperforms the proposed $O(n)$ algorithm. Investigations have shown that this is partly due to the limited symbolic simplification capabilities of the OpenModelica Compiler.

6 Discussion

6.1 Applicability

The algorithm has been tested on several different multibody models. It relies on the structural properties of the linear system as discussed in the earlier sections. Due to the fact that Tarjan's Algorithm [6] decomposes the system into separate strong connected components, the use of force elements does not influence the algorithm as long as their value does not depend on accelerations or forces in the system. Hence, Accounting

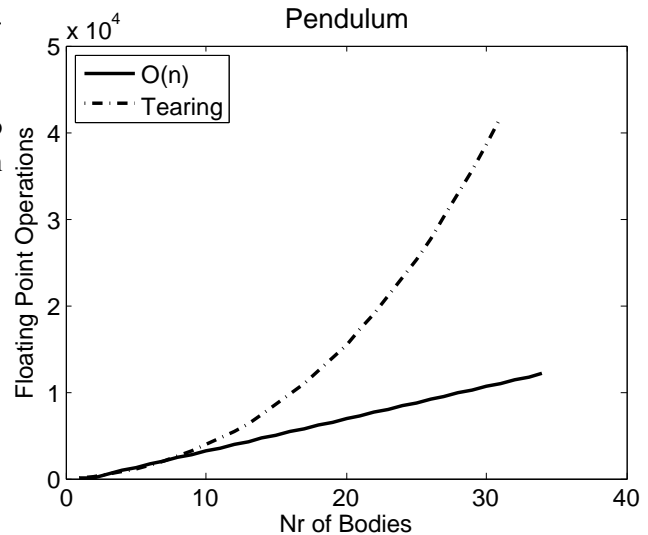


Figure 6: Results - Pendulum

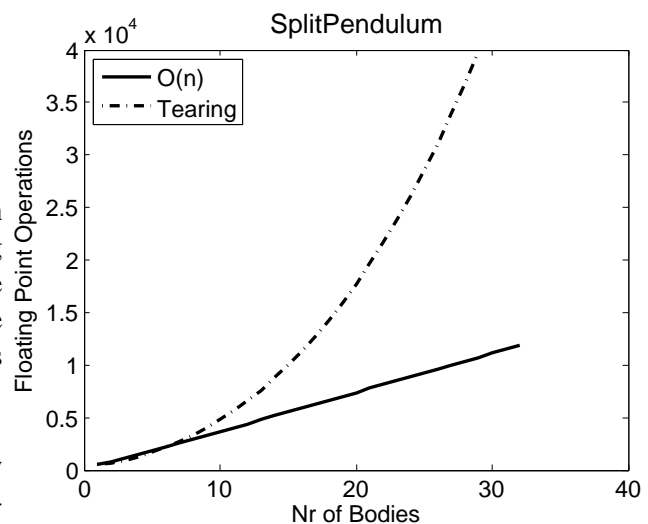


Figure 7: Results - Split Pendulum

for dry friction (tangential force depending on the normal force) for example, might cause the algorithm to fail.

Structural singularities are found during compile time, since during symbolic Gaussian Elimination each pivot is checked if it is non-zero. Problems like numerical cancellation or division by zero are not detected by the compiler and have to be reported as errors at runtime.

Due to the problems which may be encountered on some models, the algorithm should not be enabled by default. Instead it provides an interesting alternative for users who try to tune their models for faster execution times, as it would be the case in real time applications, for example.

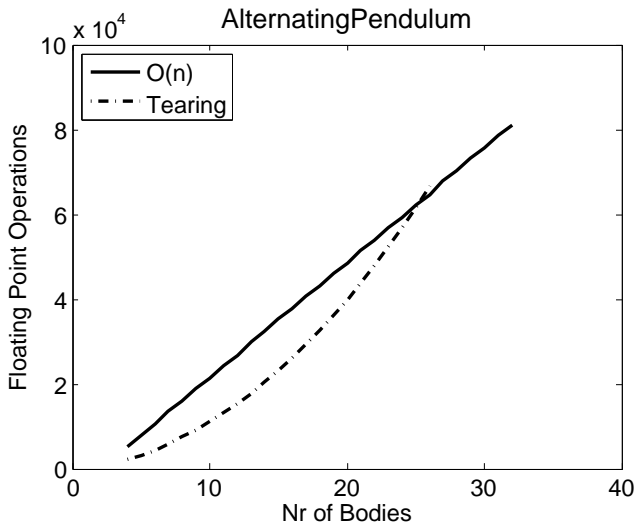


Figure 8: Results - Alternating Pendulum (3D)

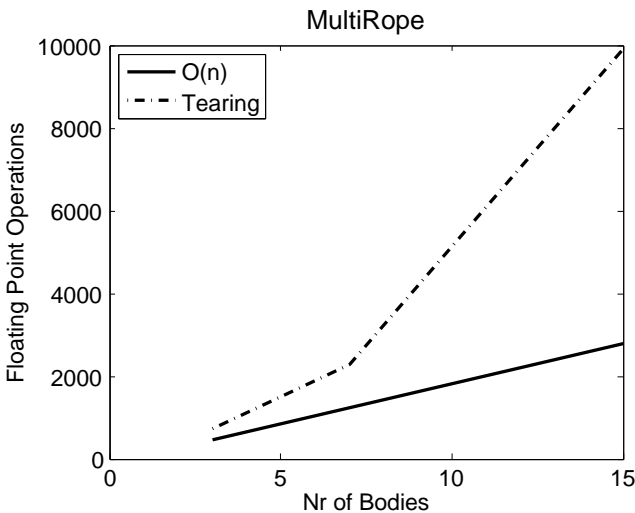


Figure 9: Results - Multi Pendulum

6.2 $O(n)$ or Tearing?

If Gaussian Elimination fails during compile time, the current implementation switches back to Tearing. However, the question arises which strong connected components should the proposed $O(n)$ algorithm be applied to. The current (presumably non-efficient) implementation is controlled by a compiler flag. If it is set, the $O(n)$ algorithm is applied to every strong connected component. Should it fail, Tearing is applied instead. A possible improvement could be, to control that either by an annotation or by comparing the operation count.

6.3 Efficiency

The investigations suggest that this algorithm indeed achieves $O(n)$ performance and the results show that it is often more efficient than Tearing. However, there is still much potential for optimization. The most promising optimization would be to exploit symmetry. This could be achieved by looking for common sub expressions.

The current version of the Modelica.Mechanics.MultiBody library however, is not suited for exploiting symmetry since all translational variables are written with respect to the world frame. Thus, for equation (1) and (3) the relationship $\mathbf{R}_{i,j} = \mathbf{R}_{j,i}^T$ does not hold. Preliminary tests have shown a 20% decrease in operation count, without the usage of a common sub expression search, when the symmetry is established by writing all translational variables with respect to the local frame_a.

7 Outlook

Next steps include adaptations to make the algorithm work reliably on models with kinematic loops. It is also worth extending the module which performs symbolic simplification by analyzing the assignments before code generation. This may also be combined with trying to exploit symmetry in order to lower the number of operations.

Lastly, it would be interesting to see if that algorithm may also be applied successfully to models from other domains, like electrical networks or chemical processes.

8 Conclusion

In this paper a special $O(n)$ algorithm for calculating the joint accelerations of a multibody system has been adapted. With the novel graph theoretic interpretation, general purpose Modelica compilers are able to solve models from Modelica.Mechanics.MultiBody with a computational effort proportional to number of bodies n compared to the usual $O(n^3)$ algorithms based on the mass matrix. A working implementation for the OpenModelica Compiler has shown a linear relationship between the operation count and degrees of freedom. When comparing the results to the tearing algorithm, it became apparent that it outperforms the proposed algorithm for non-planar models. This is partly due to the limited symbolic simplification carried out by the OpenModelica Compiler.

References

- [1] H. Elmqvist, and M. Otter: Methods for Tearing Systems of Equations in Object Oriented Modeling, Proc. ESM'94, European Simulation Multi-conference, Barcelona, Spain, June 13, 1994, pp. 326-332.
- [2] R. Featherstone: The calculation of robot dynamics using articulated-body inertias. International Journal of Robotics Research 2, May 1983, pp. 13-30
- [3] R. Featherstone: Rigid Body Dynamics Algorithms. Springer, New York, 2008
- [4] H. Brandl, R. Johanni and M. Otter: A very efficient algorithm for the simulation of robots and similar multibody systems without inversion of the mass matrix, In Kopacek, P., Troth, I. and Desoyer, K. (Eds.), Theory of Robots, Oxford, Pergamon Press, 1988, pp. 95- 100
- [5] M. Otter, H. Elmqvist, and F.E. Cellier: Relaxing: A symbolic sparse matrix method exploiting the model structure in generating efficient simulation code, Proc. Symp. Modelling, Analysis, and Simulation, CESA'96, IMACS MultiConference on Computational Engineering in Systems Applications, Lille, France, vol.1, 1995, pp. 1-12
- [6] I.S. Duff, A.M. Erisman, and J.K. Reid: Direct Methods for Sparse Matrices, Oxford University Press, 1986
- [7] S.E. Mattsson and G. Söderlind: Index Reduction in Differential Algebraic Equations Using Dummy Derivatives, SIAM Journal on Scientific Computing, Vol. 14, No. 3, pp. 677-692, 1993
- [8] R. Tarjan: Depth-first search and linear graph algorithms, 12th Annual Symposium on Switching and Automata Theory, 13-15 Oct., 1971, pp. 114 - 121
- [9] E. Carpanzano: Order Reduction of General Nonlinear DAE Systems by Automatic Tearing, Mathematical and Computer Modelling of Dynamical Systems, Vol. 6, Iss. 2, 2000

Modeling the discontinuous individual channel injection into fin-and-tube evaporators for residential air-conditioning

Martin Ryhl Kærn^{†,‡} Brian Elmegaard[†]

[†]Technical University of Denmark, Department of Mechanical Engineering
Nils Koppels Allé Bygn. 403, DK-2800 Lyngby, Denmark, e-mail: pmak@mek.dtu.dk

[‡]Former address: Danfoss A/S, Refrigeration and Air-Conditioning
Nordborgvej 81, DK-6430 Nordborg, Denmark

Abstract

In this paper a working principle based upon the novel expansion and distributor device EcoFlowTM is analyzed. The device enables compensation of flow maldistribution by control of individual channel superheat. The working principle is discontinuous liquid injection (pulsating flow) into each individual channels during a specified cycle time. Moreover, the influence of the injection cycle time is investigated together with an optional secondary flow into the other channels with regards to cooling capacity, overall UA-value and COP.

The results showed spurious fluctuations in pressure when simulating the pulsating flow, thus the dynamic behavior in the mixture two-phase flow model is insufficient to model the discontinuous liquid injection principle. Despite, the fluctuations and imperfections of the model we found that the cycle time should be kept as low as possible and that the optional secondary flow increases performance. Moreover, the paper reports on the applicability of Modelica developed models to analyze and optimize the working principle and design of expansion devices such that Modelica may be used in future development of novel discontinuous expansion devices.

Keywords: refrigeration; air-conditioning; evaporator; two-phase flow; liquid injection; pulsation; transient; dynamic; modeling; simulation; Modelica.

Nomenclature

Roman

A	cross-sectional area (m ²)
c_p	specific heat capacity (J kg ⁻¹ K ⁻¹)
C	capacitance flow (W K ⁻¹)

COP	coefficient of performance (-)
D	inner tube outer diameter (m)
d	inner tube inner diameter (m)
F_w	wall friction force (N m ⁻³)
F_o	orifice flow ratio parameter (-)
g	gravitational acceleration (m s ⁻²)
\dot{H}	enthalpy flow (W)
h	specific mixed-cup enthalpy (J kg ⁻¹)
\bar{h}	specific in situ mixture enthalpy (J kg ⁻¹)
h_{tc}	heat transfer coefficient (W m ⁻² K ⁻¹)
\dot{I}	momentum flow (N)
K	orifice flow coefficient
k	thermal conductivity (W m ⁻¹ K ⁻¹)
M	mass (kg)
\dot{m}	mass flow rate (kg s ⁻¹)
NTU	number of transfer units (-)
OD	opening degree (%)
P	channel perimeter (m)
p	pressure (Pa)
\dot{Q}	heat flow rate (W)
q_w''	wall heat flux (W m ⁻²)
R	thermal resistance (K W ⁻¹)
S	slip ratio (-)
T	temperature (K)
t	time (s)
U	velocity (m s ⁻¹)
UA	overall UA-value (W K ⁻¹)
x	vapor quality (-)
z	axial channel length (m)

Greek

α	void fraction (-)
ε	effectiveness (-)
Θ	distribution vector (-)
ρ	density (kg m ⁻³)
$\bar{\rho}$	mixture density (kg m ⁻³)

ρ'	momentum density (kg m^{-3})
θ	angle to horizontal plane (deg.)

Subscripts

ax	axial
c	condensation
cyc	cycle
e	evaporation
exp	experiment
f	saturated liquid
g	saturated gas
H	homogeneous
damp	dampening
inj	injection
rad	radial
ss	steady state
tot	total
w	wall

1 Introduction

Flow maldistribution in fin-and-tube evaporators has been shown by many investigators to reduce the performance of air-conditioning systems in terms of cooling capacity and COP. Furthermore, compensation of flow maldistribution by control of individual channel superheat has been shown to recover the penalties of flow maldistribution significantly [1, 2, 3]. Perfect control of individual channel superheats means that a thermostatic or electronic expansion valve is located on each evaporator channel and thus controls each superheat to be the same. It is not beneficial for economic reasons to install an expansion valve for each channel. Therefore, the discontinuous liquid injection principle is studied in this paper as a promising method for compensation by control of individual channel superheat. On the other hand, the tube circuitry of fin-and-tube evaporators may be optimized to compensate flow maldistribution by design [4] such that equal channel superheats occur, however, it does not ensure equal channel superheats at part-load or off-design conditions.

The focus of the current study is to gain more understanding and insight in the discontinuous liquid injection into each evaporator channels and its implications for evaporator design and system performance in terms of overall UA-value, cooling capacity and COP. We will investigate implications for two standard tube circuitries namely the face split and the interlaced evaporator, see figure 1. Especially, we strive to optimize the discontinuous liquid injection principle by study-

ing the effects of different specifications (cycle time and optional secondary flow) and provide guidelines for optimal energy efficiency. For simplicity we do not consider actual flow maldistribution when evaluating the effect of cycle time and optional secondary flow. The injection principle is essentially two-phase flow pulsations and the study may show the potential of increasing capacity and COP by employing pulsations to the flow.

The modeling of the liquid injection dynamics showed spurious fluctuations in pressure, which have not been observed as high in any similar experiments carried out at Danfoss. The current analysis should therefore be seen as a first study of the injection dynamics with the current model approach and limitations. When simulating the injection dynamics, we must keep in mind that the correlations for heat transfer, friction and void may become invalid at large transients in mass flow, since they are developed from steady state experiments. Furthermore, the discontinuous refrigerant injection is essentially pulsating two-phase flow, and the significance of the liquid/vapor interfacial dynamics may become important such as interfacial friction and drag and/or thermodynamic non-equilibrium effects. These phenomena are not included in the typical mixture two-phase flow model used in many Modelica libraries, and also used in the current study (developed in Kærn [3]).

1.1 Liquid injection principle

The liquid injection principle is based on the recently developed Danfoss product (EcoFlowTM [5]). Actually, the EcoFlow valve does not measure the individual channel superheats but only the overall superheat. Furthermore, it does not provide continuous refrigerant flow in each channel, but rather discontinuous individual channel injection (modulation of each channel flow) with optional secondary flow to the other channels. The optimal distribution of mass flow rate (at flow maldistribution) is then found from a distribution analysis performed at specific time intervals during operation, see Mader and Thybo [6]. The distribution analysis is essentially carried out by control algorithms, where the importance of each individual channel on the overall superheat is measured in order to find the optimal distribution. The individual channel superheats become the same at the optimal mass flow distribution.

The individual injection is performed by a stepper motor (48 steps per revolution), which rotates the distributor disc, see figure 2a. The EcoFlow valve comes

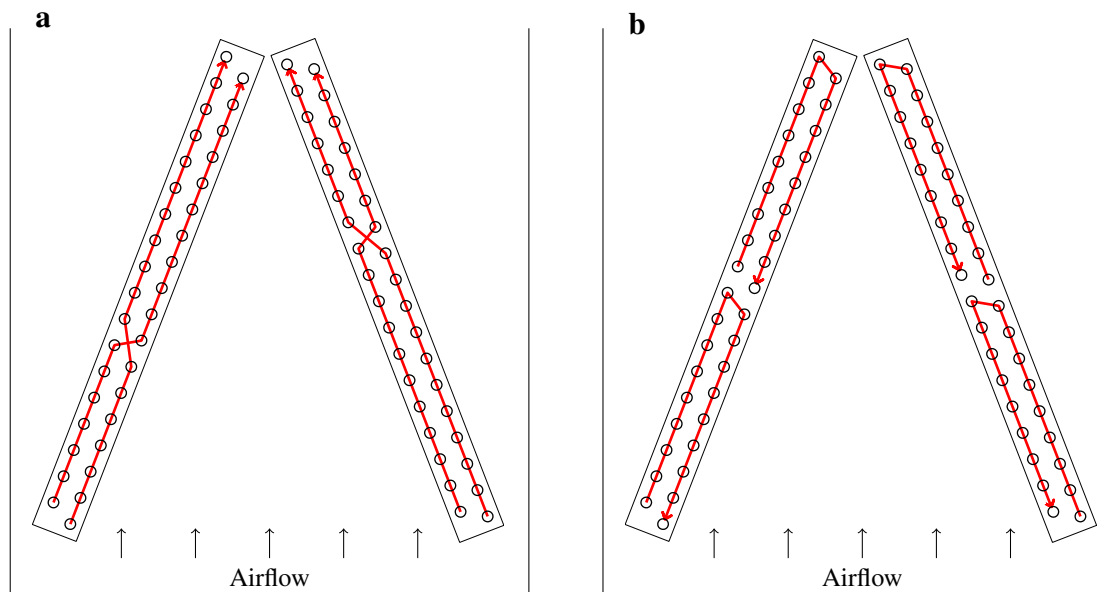


Figure 1: Tube circuitries of the interlaced evaporator (a) and the face split evaporator (b).

in two different designs, i.e. a multi-orifice (MO) design (main orifice + secondary orifices) and a single-orifice (SO) design (main orifice only), see figure 2b and 2c. The orifice size of the SO design is larger, since more refrigerant needs to pass through the main orifice. The SO design enables the possibility of individual channel defrost during cooling operation (no defrost periods) for the face split evaporator only, see figure 1b. As we shall see later, the results show that the performance in steady state without considering frost build-up becomes a bit smaller when using the SO concept. Furthermore, all orifices of both designs are closed in between each channel injection.

1.2 Objectives and content

The first objective is to evaluate the effect of the cycle time for the MO and SO design concepts, i.e. the time it takes for one revolution. The second objective is to evaluate the size of the secondary orifices in the MO design compared to the main orifice. The questions that are sought to be answered are:

- What is the minimum cycle time for discontinuous liquid injection? Too large cycle times will cause too much dry-out of the channels.
- Does capacity decrease or increase by the discontinuous liquid injection (pulsating flow)?
- How much refrigerant should pass through the main and secondary orifices in the MO design?

Note that the results is focused on the steady state performance in terms of overall UA-value, cooling ca-

capacity and COP, where the dynamics of the refrigerant injection is modeled.

The paper starts by a brief description of the liquid injection modeling and use of experimental results for evaluating orifice flow coefficients for the actual MO and SO designs. Then the pressure fluctuations caused by the liquid injection modeling is considered and compared to experiments using an earlier MO design and performed on the interlaced tube circuitry. Finally, the effect of the cycle time and flow ratio between main and secondary orifices of the MO concept are investigated.

2 Modeling approach

This section describes the model that was implemented in the Modelica language of the discontinuous liquid injection principle. Furthermore, the system model is described with focus on the evaporator.

2.1 Injection modeling

This section describes the experimental data reduction that was performed of actual EcoFlow capacity tests, in order to obtain the orifice flow coefficients for both MO and SO designs (see figure 2). The goal of the data reduction is to compute the mass flow through the main orifice and secondary orifices at different pressure levels and opening degrees (when the expansion valve is open only). The capacity tests provide continuous capacity (evaporation of refrigerant) or mass

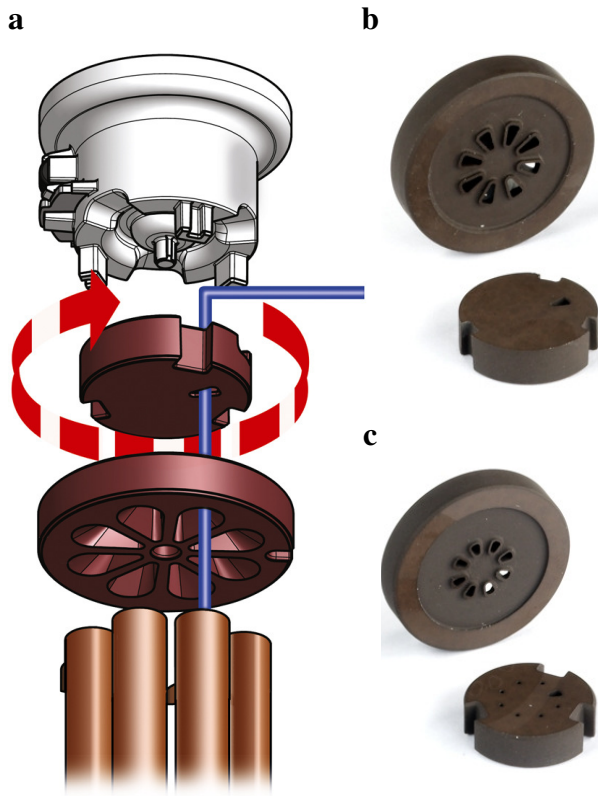


Figure 2: EcoFlow distribution method and refrigerant flow through discs (a), single-orifice (SO) discs (b) and multi-orifice (MO) discs (c).

flow rate through the valve, but we are only interested in the mass flow through the valve when it is open. When knowing the orifice flow coefficient K , the mass flow through the valve may be computed by the single phase orifice equation as

$$\dot{m}_{\text{open}} = KA\sqrt{2\rho_f(p_{\text{in}} - p_{\text{out}})} \quad (1)$$

where A is the flow area of the orifice, ρ_f is the saturated liquid density, p_{in} and p_{out} are the pressure at inlet and outlet of the valve. The use of equation 1 is the standard method of developing empirical equations to predict mass flow rate through orifices [7] even in refrigerant expansion devices [8, 9].

Two-phase flow effects such as partial vaporization (flashing) are included in the flow coefficient. Furthermore, the capacity tests of the orifice discs were only carried out at standard conditions. It means that K will not be dependent on the pressure levels, and is thus assumed to be constant at different pressure levels. The standard conditions for these capacity tests are: Evaporation at 5°C , condensation at 32°C , 4 K subcooling and no superheat. The relation between the experimental mass flow rate and valve capacity is thus

$$\dot{Q}_{\text{exp}} = \dot{m}_{\text{exp}}[h_g(p_{\text{out}}) - h(p_{\text{in}}, T_{\text{in}})] \quad (2)$$

The stepper motor has 48 steps per revolution equaling 7.5 degree rotation per step. The step time is 10 ms per step, i.e. a minimum of 480 ms per revolution (minimum cycle time). Due to the opening and closing of the valve, the liquid refrigerant before the valve will create a *fluid hammer* (also called a *hydraulic shock*). The moving liquid is suddenly forced to stop, and the pressure builds up before the valve and a pressure wave will propagate upstream. In order to eliminate the peak forces acting on the valve, the speed of the stepper motor is dampened as the valve opens and closes.

To find the actual mass flow through the valve when open we need to know the opening time of the valve (injection time). The actual injection time is a function of cycle time, opening degree, damping time and step time of the stepper motor. A detailed description is given in Kærn [3], however, it is simply a matter of tracking the time when open and closed. When the injection time is known the mass flow through the valve when open may be computed by mass continuity as

$$\dot{m}_{\text{open}} = \dot{m}_{\text{exp}} \frac{t_{\text{cyc}}}{t_{\text{inj}}} \quad (3)$$

and used in equation 1 to compute the flow coefficient K for the total flow through main and secondary orifices. The flow coefficient is thus for actual design and number of discharge channels (EcoFlow is made with up to 8 discharge channels), and is a function of opening degree, cycle time, step time and damping time.

In this paper we only consider four channel evaporators, i.e. two coils with two channels each. Therefore, the flow coefficients were only computed on the four channel orifice discs with MO and SO designs. The standard EcoFlow time settings are a step time of 10 ms and a damping time of 120 ms for both opening and closing. Using the capacity tests, we computed the flow coefficients for the total flow as function of opening degree for cycle times 6, 10 and 20 seconds for both MO and SO designs. For the SO design the total flow comes through the main orifice, however, for the MO design we need additional information on how much flow that goes into the main and secondary orifices, respectively.

Fortunately, a capacity test was also performed at steady state conditions, i.e. no rotation of the distributor disc and fully open continuous flow. The test was done at all orifices open, but also at main orifice closed, which gives us the *flow ratio parameter* be-

tween the main orifice flow and total flow in steady state as

$$F_o = \frac{\dot{m}_{\text{main,ss}}}{\dot{m}_{\text{tot,ss}}} = 0.492 \quad (4)$$

The ratio is assumed to be independent of the cycle time and damping time, and thus directly used to distribute the total mass flow to the main and the secondary orifices when the valve is open. The total mass flow when the valve is open and the corresponding steady state mass flow are shown on figure 3a. Figure 3b shows the corresponding flow coefficients.

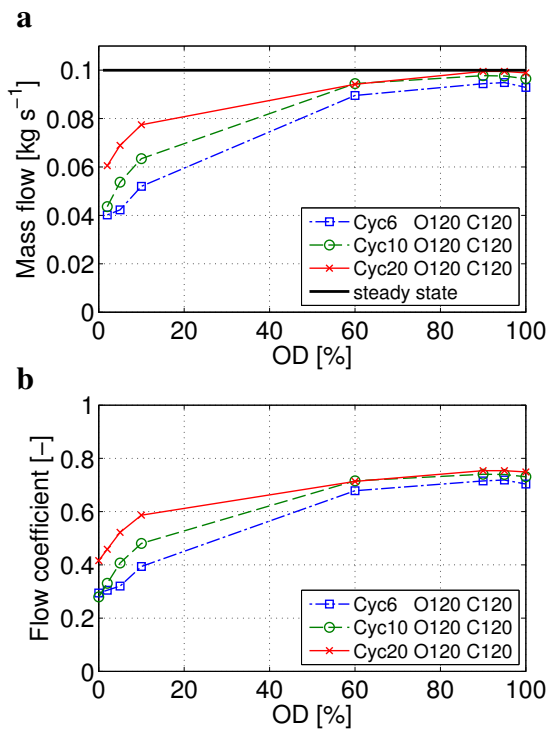


Figure 3: Total mass flow rate when valve is open (a) and flow coefficients (b) as function of opening degree (MO design); Cyc=cycle time [s], O=opening damping time [ms], C=closing damping time [ms].

We assume that the accelerational effects of the fluid at opening and closing may cause the differences in the flow coefficients and mass flows, which tends to differ more at low opening degree, where the accelerational effects should play a larger role compared to the actual mass transferred through the valve. As expected, the mass flow curves are below the steady state mass flow and becomes closer at high opening degree, where the opening and closing have smaller influence. Unfortunately, there were no measurements between 10% and 60% opening degree.

The expansion process may experience choking of

the flow, i.e. the mass flow may not increase by decreasing the downstream pressure and is only a function of upstream conditions. Using the above modeling approach does not include the choking phenomenon and the mass flow is essentially a function of pressure difference and flow coefficient. It is thus assumed that choking of the flow is not existing.

2.1.1 Implementation

The implementation of the liquid injection model in Modelica is done by using the CombiTable1D model from the Standard Modelica Library, i.e. one-dimensional linear table interpolation of the flow coefficients. The mass flow rates through the main orifice and secondary orifices (MO) are then computed using equation 1 and 4. Now it is just a matter of computing the *individual* channel opening and closing time during each cycle. A distribution vector is defined as

$$\sum_{i=1}^N \Theta_i = 1 \quad (5)$$

which determines the time period associated with each channel $t_{\text{tube},i}$ as

$$t_{\text{tube},i} = t_{\text{cyc}} \Theta_i \quad (6)$$

where i denotes the channel number and N the total number of channels. The injection time for each channel is computed by

$$t_{\text{inj},i} = \left(t_{\text{tube},i} - \frac{N_{\text{step}} t_{\text{step}}}{N} - \frac{N_{\text{damp}} t_{\text{damp}}}{N} \right) \frac{\text{OD}}{100} + \frac{N_{\text{damp}} t_{\text{damp}}}{N} \quad (7)$$

The first term in the parenthesis is the controllable time per channel (minimum cycle time subtracted) times opening degree. The second term counts for the additional mass flow that would occur even though the opening degree is zero. The dampening time occurs from approximately 70% to 100% opening area of the orifice (as the disc turn). For simplicity, the additional mass flow is assumed to be the mass flow when fully open times the damping time.

The opening of each channel is assumed to occur at $t_{\text{tube},i}/2 - t_{\text{inj},i}/2$. The closing is then at $t_{\text{tube},i}/2 + t_{\text{inj},i}/2$. The changes in mass flow rate are made smooth by use of the first order continuous functions as described in [3, 10] for numerical reasons. The transition time was chosen to be 0.1 seconds.

Figure 4 shows some examples of the MO liquid injection model at a cycle time of 10 seconds. It illustrates the working principle of the liquid injection

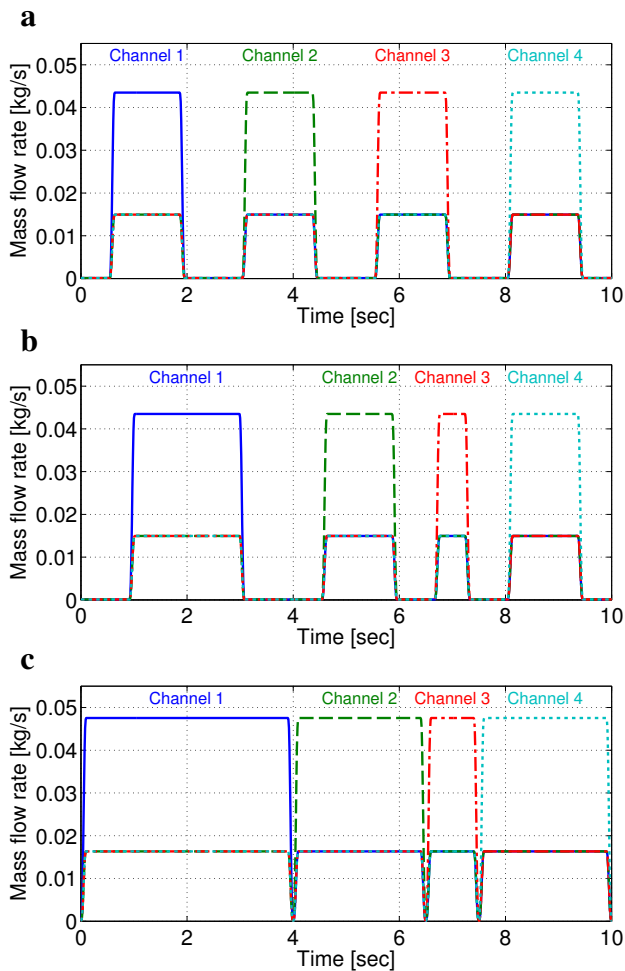


Figure 4: Mass flow distributions for liquid injection model with MO design at cyclotime = 10 s; $\Theta = [0.25, 0.25, 0.25, 0.25]$, OD = 50% (a); $\Theta = [0.4, 0.25, 0.1, 0.25]$, OD = 50% (b); $\Theta = [0.4, 0.25, 0.1, 0.25]$, OD = 100% (c); $p_e = 9.3$ bar and $p_c = 19.8$ bar (standard condition).

model as the opening degree and the distribution vector are changed. Throughout this paper we do not consider compensation of flow maldistribution, thus the liquid injection model runs in *even flow* mode (figure 4a) and the distribution vector becomes $\Theta = [0.25, 0.25, 0.25, 0.25]$. In *compensating flow* mode the values in the distribution vector need be controlled in the numerical model according to the individual channel superheat.

2.2 Model setup

The numerical model is described in Kærn et al. [11] for a co-axial evaporator and has been updated as described in Kærn [3] to model the full system (condenser and compressor also) and the tube circuitries

of the interlaced and face split evaporators, see figure 1. The model is implemented in the Modelica language and Dymola 7.4 [12] is used as simulator. The Modelica language facilitates object-oriented programming, which is important for model reuse and extension. Dymola has been well tested within the field of air-conditioning and refrigeration [13, 10]. Thermophysical properties for R410A are obtained from the Refeqns package [14]. In order to model the refrigerant distribution and circuitry in the evaporator a dynamic distributed one-dimensional mixture finite volume model was chosen. For the condenser, the simpler moving boundary model of Zhang and Zhang [15] was chosen, which averages the vapor, two-phase and liquid regions. The models of the expansion and compressor are quasi-static. Momentum transfer and frictional pressure drop are only addressed in the evaporator tubes, U-bends and feeder tubes, in order to predict the mass flow distribution in the evaporator. Furthermore, the void fraction model by Zivi (1964) is used to model the refrigerant charge of both condenser and evaporator.

Since the evaporator pressure showed spurious fluctuations when simulating the injection principle, we included the refrigerant flow equations and implementation for the evaporator model in the appendix such that these may be studied by the reader. Furthermore, we did not use the Modelica stream prefix. Since the compressor runs at constant speed, we did not observe flow reversal during the flow pulsations.

2.2.1 Geometry and correlations

Table 1 shows the main geometry of the test case evaporator and condenser. The tube inner walls are smooth. Furthermore, the feeder tubes to the evaporator have an internal diameter of 3 mm and a length of 300 mm. The manifold inner and outer diameter is 16 mm and 19 mm, respectively, and its length is 5 m from the evaporator to the compressor. Note that the coil geometry is the same for both the interlaced and face split evaporator, however, the tube connections or circuiting are different as shown on figure 1. Furthermore, the simulation of the injection is very CPU demanding and for this reason we chose to use only one cell per tube for the evaporator. In terms of convergence in total cooling capacity of the evaporator, this choice is within 2% of the total cooling capacity at 5 cells per tube [3]. In the condenser, refrigerant enters four channels and is mixed before entering a fifth channel. Since the circuitry is not addressed in the condenser, it is assumed to be four straight tubes.

Table 1: Main geometry of evaporator and condenser

	Evaporator	Condenser
Number of coils	2	1
Number of channels in each coil	2	5
Number of tubes in each channel	18	6
Tube length [mm]	444.5	2100
Inner tube diameter [mm]	7.6	7.6
Outer tube diameter [mm]	9.6	9.6
Transverse tube pitch [mm]	25.4	25
Longitudinal tube pitch [mm]	21.25	
Fins	Louvred	Louvred
Fin pitch [mm]	1.81	1.15
Total outside area [m ²]	17.3	52.2
Number of cells per tube	1	

Table 2: Overview of used correlations

Air-side	
Heat transfer	Wang et al. (1999)
Fin efficiency	Schmidt approximation (1949)
Single-phase	
Heat transfer	Gnielinski (1976)
Friction	Blasius (1913)
Bend friction	Ito (1960)
Two-phase	
Heat transfer (evaporator)	Shah (1982)
Heat transfer (condenser)	Shah (1979)
Void fraction	Zivi (1964)
Friction	Müller-Steinhagen and Heck (1986)
Bend friction	Geary (1975)

Full references are given in Kærn [3].

Each discrete cell of the evaporator is calculated as a separate heat exchanger with uniform transport properties. Mass, momentum and energy conservation equations are applied to the refrigerant in each cell, where thermodynamic equilibrium is assumed. Furthermore, changes in kinetic and potential energies are neglected. It is assumed that the tube walls have rotational symmetry (no azimuthal heat conduction) and negligible axial heat conduction. Mass and energy conservation equations are applied to the air, which is assumed to be dry. Similar assumptions are used in the condenser model of the refrigerant and airflow, however the heat resistance and the dynamics in the condenser wall are neglected. The used correlations for both the evaporator and the condenser are given in table 2. Furthermore, effectiveness-NTU relations are employed.

The expansion process is modeled as an isenthalpic process and the opening degree from equation 7 essentially controls the superheat out of the evaporator. The manifold is modeled by mixing of the refrigerant

streams, i.e. mass and energy conservation equations are applied. The dynamics of the manifold wall is included and heat transfer is modeled using a constant heat transfer coefficient of $700 \text{ Wm}^{-2}\text{K}^{-1}$. The geometric volume flow of the compressor is $6.239 \text{ m}^3\text{h}^{-1}$, and polynomials from the rating of the compressor are used to compute the isentropic and volumetric efficiencies.

2.2.2 Boundary conditions

The liquid injection model controls the overall superheat to 5 K by the opening degree using a PI-controller. During start-up of the simulation, the charge of the system is determined so that the subcooling becomes 2 K. The indoor and outdoor air temperatures are 26.7°C and 35°C , respectively. The mean frontal air velocities are 1.16 and 0.68 m s^{-1} to the evaporator and condenser, respectively.

3 Experimental comparison

In this section we compare the injection modeling with experiments carried out at Danfoss Nordborg. The dynamic behavior observed in the simulations showed fluctuations in important variables such as superheat and evaporating pressure. In Kærn [3] a sensitivity analysis of the fluctuations were performed in order to better understand the causes of the fluctuations, however, sensible variables such as void fraction and manifold+suction volume did not eliminate the fluctuations satisfactorily.

The fluctuations in the model have a time period corresponding to the cycle time of the liquid injection model divided by the number of channels in the evaporator (for *even flow* mode, see figure 4a). These fluctuations have not been observed as high in any experiments carried out at Danfoss, where the sampling frequency has been high enough to capture these fluctuations. The sampling frequency is often chosen to be 1 s^{-1} for refrigerant temperature and pressure measurements at Danfoss, which is too low for capturing the injection dynamics seen in the numerical model.

3.1 The experiments

The EcoFlow experiments were performed on a bit different system and conditions than described in previous section. The system comprised a 10.5 kW interlaced evaporator, a hermetic scroll compressor, micro-channel condenser and an early MO disc version. The early design of the MO disc is estimated to have a flow

Table 3: Reduced experimental boundary conditions

Superheat	5 K
Pressure out of condenser	31.9 bar
Liquid temperature out of condenser	45.6 °C
Volume flow out of evaporator	7.17 m ³ h ⁻¹
Indoor air temperature	24.3 °C
Indoor frontal air velocity	2.98 m s ⁻¹

ratio parameter F_o (equation 4) of 0.8, which reflects the earlier version cross-sectional areas of the main and secondary orifices. Furthermore, the flow coefficients, the step time and damping time are assumed to be the same as the final MO disc design. The cycle time was six seconds in the experiments and the flow distribution mode was *even flow*, see figure 4a.

These experiments are the most recent experiments carried out at Danfoss in Nordborg on a fin-and-tube four channel evaporator using the EcoFlow valve. Later experiments including compensation were performed with the final EcoFlow version, however, on larger capacity units with six or eight channels each, which complicates the simulations drastically. For these reasons, the earlier EcoFlow MO experiments were chosen for the comparison. More information about the experimental data is given in Kærn [3]. The experimental data is reduced in order to be used as input to the evaporator model only, thus we only simulate the 10.5 kW evaporator and manifold+suction volume in this comparison. Table 3 lists the model inputs.

Figure 5a and 5b show the experimental superheat and pressure fluctuations during three cycles. The corresponding model results are shown in figure 5c and 5d. Note that the thick curve around 5 K is overall superheat. Furthermore, the experiments show a bit higher individual superheats. This is because that they were measured on the tube wall surface with insulation around the tube, and may have heat entering from the surroundings.

When comparing to the experimental data, it is seen that the pressure fluctuations are smaller (approximately one third in amplitude of the numerical results). It is difficult to make this conclusion based on these experimental results, since the sample time was only 1 s⁻¹ for the pressure. However, the experiments carried out at Danfoss with higher frequency did not show as high fluctuations as the numerical model does here. The reason for these high fluctuations in the numerical model have not been obtained so far. However, we believe that the interfacial dynamics of the two-phase flow and the presence of thermodynamic non-equilibrium may be responsible for the dampening of

the pressure fluctuations in the experiments. These are inherently exclusive in the mixture two-phase flow model. In addition, the refrigerant heat transfer, pressure drop and void correlations are developed from steady state experiments and employed at large transients, however, no dynamic two-phase flow correlations (pulsating flow) were found in the literature.

If we compare the individual superheat measurements and the prediction by the numerical model, then the accordance is much more acceptable. Both the measurements and the model predictions show the effect of the liquid injection into each channel, since they fluctuate similarly at a time period corresponding to the cycle time. Furthermore, the superheat decreases as the refrigerant enters through the main orifice into each channel as indicated on figure 5e. The corresponding mass inside each channel is shown on figure 5f, which increases when the refrigerant enters through the main orifice and otherwise decreases.

What is probably most important is the individual channel overall UA-value in figure 5g, which shows a decrease just before new refrigerant is fed to the corresponding channel. There may be an optimization potential here if the cycle time is chosen such that the UA-value decrease is avoided. Figure 5h shows the corresponding individual channel pressure drop by friction and acceleration due to density and mass flux differences. When considering the individual channel pressure drop due to friction and acceleration, one may expect that this is the cause of the pressure fluctuations, however, the sensitivity analysis from [3] proves otherwise. It is interesting to note that the accelerational pressure drop is positive as the refrigerant is fed to each channel. This is because the refrigerant mass flow is higher at the inlet compared to the outlet of the channel, i.e. the difference in momentum flow between inlet and outlet is positive.

4 Simulation results

Despite the presence of the pressure fluctuations, the numerical model is used to perform simulations of the significance of the cycle time for both the multi-orifice (MO) and single-orifice (SO) designs. Furthermore, the flow ratio parameter F_o (equation 4) for the MO design will be investigated, i.e. the flow distribution between the main and secondary orifices of the MO design.

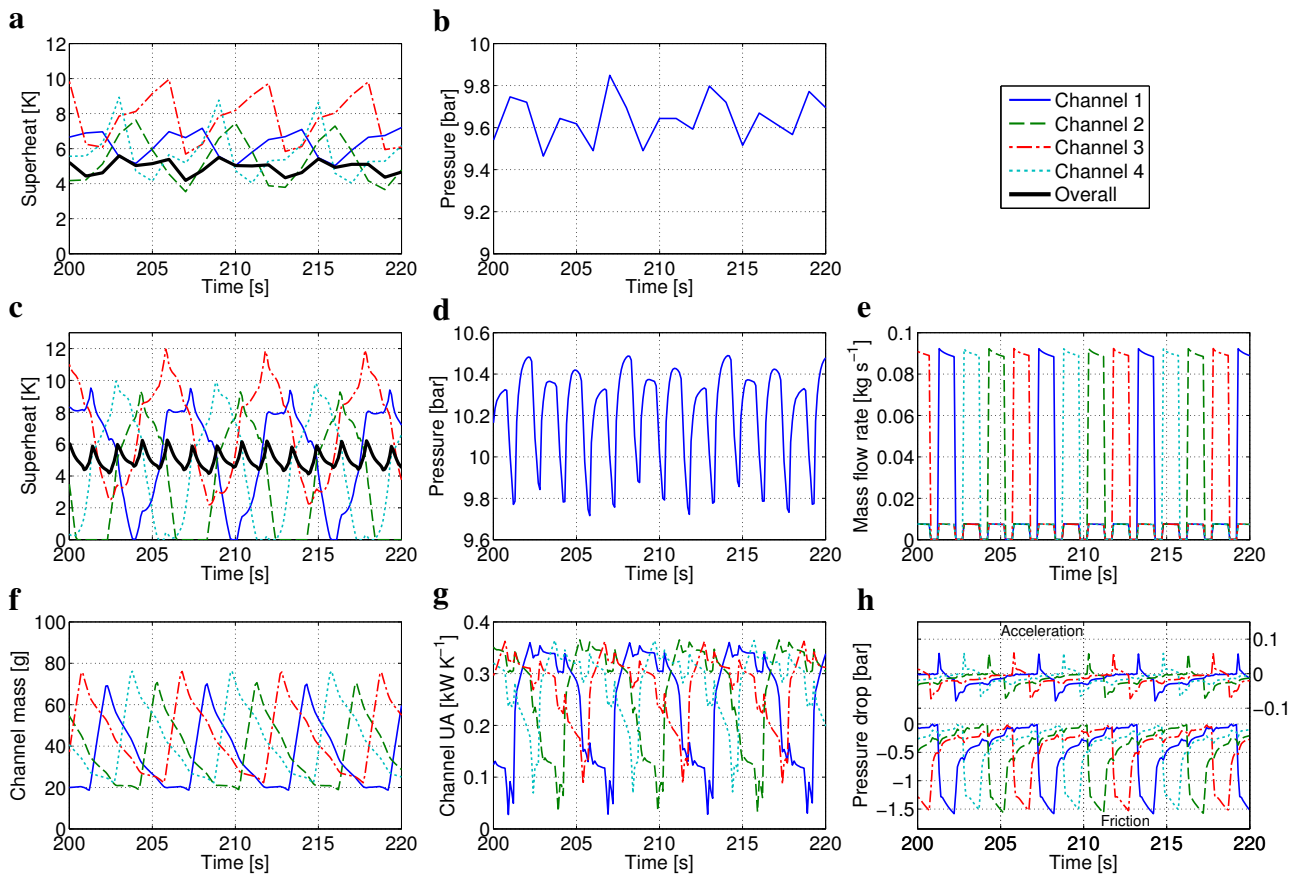


Figure 5: Zoomed-in experimental superheats and suction pressure (a,b); Model comparison (c,d) and other model results (e-h) at uniform airflow: Inlet individual channel mass flow rate (e), individual channel mass (f), individual channel overall UA-value (g) and individual channel accelerational and frictional pressure drop (h).

4.1 Cycle time

Figure 6 (a,b,c) shows the UA-value, cooling capacity and COP using MO and SO designs as function of the cycle time. Note that the orifice flow coefficients for the 3 second cycle time simulations were assumed to be the same as for the 6 second cycle time case.

The results show that the MO design performs better than the SO design. Furthermore, the cycle time should be kept as low as possible. If flow pulsations increase heat transfer we would have expected an optimum cycle time, but it seems to be outside the cycle times considered or not shown using the current mixture model and limitations (see discussion). The simulation using the SO design at a cycle time of 20 seconds failed and was not obtainable. It also seems that this case decreases the performance drastically. The question regarding which cycle time is the maximum limit is difficult to answer based on the present results. For these four channel evaporators it seems that the maximum cycle time is 10 and 6 seconds for the MO and SO design, respectively. Otherwise, the channels dry-out too much when the valve is closed.

The face split circuitry shows the best performance in contrast to the interlaced circuitry at uniform flow conditions for each distribution method. This is because the superheated regions of the face split evaporator is in the first tube row and is thus minimized. This also means that the face split evaporator performs better than the interlaced if flow maldistribution is compensated as also shown by Kærn [3].

4.2 MO flow ratio

Figure 6 (d,e,f) shows the UA-value, cooling capacity and COP as function of the flow ratio parameter F_o , and at a cycle time of 6 seconds. It shows that the maximum performance is when F_o equals 0.25, which means that the main and secondary orifices have the same dimension, thus no possibility to distribute mass individually. Essentially, all the curves on figure 4a coincides, i.e. the flow is distributed evenly to all orifices at each injection.

It shows that for uniform flow conditions, the optimal refrigerant mass flow distribution is uniform. However, the decrease in performance as F_o increases

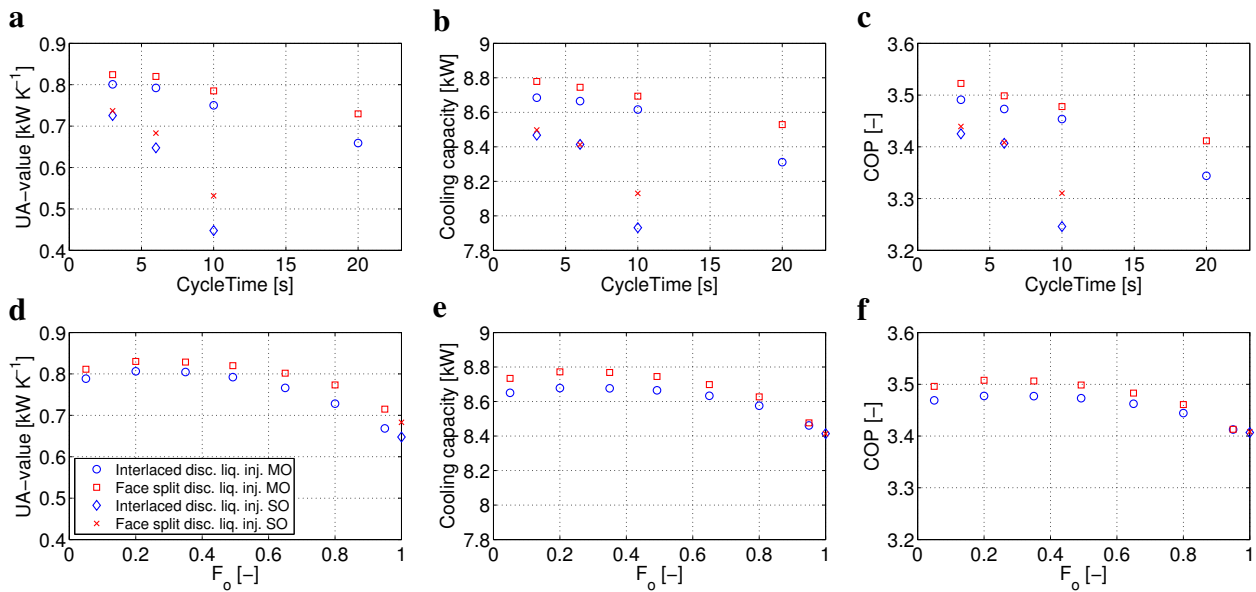


Figure 6: UA-value, cooling capacity and COP vs. the cycle time (a,b,c) at $F_o = 0.492$ for the MO design; UA-value, cooling capacity and COP vs. the flow ratio parameter (d,e,f) at cycle time $t_{cyc} = 6$ seconds.

is small and the maximum limit seems to be around 0.6. Otherwise the secondary channels will also dry-out too much. Furthermore, the $F_o = 95\%$ results of the MO design seems to be close to the SO design results presented here at $F_o = 100\%$.

5 Discussion

It is difficult to claim whether the two-phase flow pulsations increase or decrease the heat transfer mechanism. Firstly, the two-phase flow regimes are broken up by the flow pulsations and giving rise to new discontinuous flow patterns, which are not properly reflected in the steady state correlations for refrigerant heat transfer, pressure drop and void fraction. No two-phase flow correlations were found in the literature by the authors that were developed for discontinuous liquid injection or pulsating flow. Secondly, the mixture two-phase flow model (also used in many Modelica libraries) showed spurious pressure fluctuations, which have not been observed as high in any experiments carried out at Danfoss. The amplitude of the fluctuations are approximately 3 times higher in the model compared to similar experiments. Thus the readers need to be cautioned that the results and conclusions from the liquid injection modeling are obtained despite the presence of these fluctuations. It is believed that the absence of the two-phase interfacial dynamics in the mixture two-phase flow model is the main cause of the high pressure fluctuations.

It needs to be stressed that it is not the finite volume

model approach itself that leads to these fluctuations, but rather the governing phasic equations when added and becoming mixture equations. The model could be a separated flow model that includes the governing phasic equations and possibly the finite volume model could be used to discretize the phasic equations again. It is difficult to claim what may minimize the pressure fluctuations. The only separated flow model known to the authors that is implemented in Modelica is the work of Bauer [16], who implemented both phasic momentum equations. It resulted in another state variable (the velocity difference between the phases), which essentially is related to the void fraction. It would be interesting to look deeper into such model approaches when considering these fluctuations. Similarly, more dedicated experimental evidence of these fluctuations would be interesting to have.

6 Conclusion

We conclude that the typical mixture two-phase flow model that is used in many Modelica libraries is insufficient to model the discontinuous liquid injection principle (pulsating flow) into each evaporator channel. This is because the simulations showed spurious fluctuations in evaporating pressure and superheats, which have not been observed as high in any experiments carried out at Danfoss. Furthermore, it should be stressed that the correlations for heat transfer, pressure drop and void fraction employed in mixture two-phase flow models do not reflect the dynamic behavior

of the pulsating flow, since they are based upon steady state experiments. To draw detailed conclusions, further studies on the discontinuous liquid injection principle should be conducted in order to fully understand and model the phenomenon.

Despite the fluctuations, two orifice designs of the discontinuous liquid injection principle were investigated in uniform flow conditions, i.e. the multi-orifice (MO) design and the single-orifice (SO) design. The multi-orifice design allows for a secondary flow into the remaining channels at each channel injection.

The simulations of the discontinuous liquid injection principle showed that the MO design gave better performance compared to the SO design, without considering the possible individual channel defrost possibility of the SO design for the face split circuitry. In addition, the main flow and the individual secondary flows in the MO design should be kept as even as possible while having the required mass flow distribution control band. Based upon the four channel evaporator that were analyzed, it is recommended that the cycle time should be kept below 10 and 6 seconds for the MO and SO designs, respectively. Furthermore, the flow ratio parameter should be around 0.6, or adapted to specific tube circuitry according to the required mass flow distribution control band.

A Refrigerant flow equations and implementation (evaporator model)

This appendix describes the refrigerant flow equations and implementation for the evaporator model only. It is done in order to fully state the equations that lead to the spurious fluctuations in evaporating pressure when simulating the liquid injection principle.

A.1 Mixture two-phase flow

The model of the one-dimensional two-phase flow is the simplest form, i.e. the mixture model as derived by performing a differential analysis on each phase and adding the phasic equations [17]. The result is the mixture mass conservation, the mixture momentum conservation and the mixture energy conservation equations given by

$$A \frac{\partial \bar{\rho}}{\partial t} + \frac{\partial \dot{m}}{\partial z} = 0 \quad (8)$$

$$\frac{\partial \dot{m}}{\partial t} + \frac{\partial}{\partial z} \left(\frac{\dot{m}^2}{\rho' A} \right) = -A \frac{\partial p}{\partial z} - F_w A - \bar{\rho} g A \sin \theta \quad (9)$$

$$A \frac{\partial}{\partial t} (\bar{\rho} \bar{h} - p) + \frac{\partial}{\partial z} (\dot{m} h) = P q_w'' \quad (10)$$

where it has been assumed that thermodynamic equilibrium exists and that the changes in kinetic and potential energy are negligible. The mixture density, specific in situ enthalpy, specific mixed-cup enthalpy and momentum density are given by

$$\bar{\rho} = \rho_g \alpha + \rho_f (1 - \alpha) \quad (11)$$

$$\bar{h} = [\rho_f h_f (1 - \alpha) + \rho_g h_g \alpha] / \bar{\rho} \quad (12)$$

$$h = (1 - x) h_f + x h_g \quad (13)$$

$$\rho' = \left(\frac{(1 - x)^2}{\rho_f (1 - \alpha)} + \frac{x^2}{\rho_g \alpha} \right)^{-1} \quad (14)$$

where the void fraction is defined as $\alpha = A_g/A$, and the vapor quality is defined as $x = \dot{m}_g/\dot{m}$.

Using the definition of the slip ratio, the void fraction and the vapor quality, the fundamental void-quality relation can be derived as

$$\begin{aligned} S &= \frac{U_g}{U_f} = \frac{\frac{\dot{m}_g}{\rho_g \alpha A}}{\frac{\dot{m}_f}{\rho_f (1 - \alpha) A}} \\ &= \frac{x}{1 - x} \frac{\rho_f}{\rho_g} \frac{1 - \alpha}{\alpha} \end{aligned} \quad (15)$$

and rewritten in terms of the void fraction as

$$\alpha = \left[1 + \frac{\rho_g}{\rho_f} \frac{1 - x}{x} S \right]^{-1} \quad (16)$$

If homogeneous flow is assumed, then $S = 1$ and the homogeneous void fraction, α_H , may be calculated by equation 16. Furthermore, for homogeneous flow it can be shown that $\bar{h} = h$ and $\rho' = \bar{\rho} = \rho_H$ by using the homogeneous void fraction, where the homogeneous mixture density, ρ_H , becomes

$$\rho_H = \left(\frac{x}{\rho_g} + \frac{1 - x}{\rho_f} \right)^{-1} \quad (17)$$

The state variables are chosen to be \bar{h} and p . The derivative of the mixture density with respect to time is computed by use of the chain rule

$$\frac{\partial \bar{\rho}}{\partial t} = \frac{\partial \bar{\rho}}{\partial p} \bigg|_{\bar{h}} \frac{\partial p}{\partial t} + \frac{\partial \bar{\rho}}{\partial \bar{h}} \bigg|_p \frac{\partial \bar{h}}{\partial t} \quad (18)$$

where the partial derivatives of mixture density with respect to pressure and in situ enthalpy are calculated by numerical finite difference as

$$\left. \frac{\partial \bar{\rho}}{\partial p} \right|_{\bar{h}} = \frac{\bar{\rho}(p + \Delta p, \bar{h}) - \bar{\rho}(p, \bar{h})}{\Delta p} \quad (19)$$

$$\left. \frac{\partial \bar{\rho}}{\partial \bar{h}} \right|_p = \frac{\bar{\rho}(p, \bar{h} + \Delta \bar{h}) - \bar{\rho}(p, \bar{h})}{\Delta \bar{h}} \quad (20)$$

Equations 8, 9 and 10 are discretized according to the Finite Volume Method (FVM), where the number of control volumes must be high enough to resolve the spatial distribution of properties.

The staggered grid structure is adopted as described by Patankar [18]. It means that the mass and energy conservation will be solved on the control volume grid, and the momentum equation will be solved on a staggered grid as depicted on figure 7, where ψ denotes a thermodynamic quantity and $\hat{\psi}$ its approximation. Similar discretization methodology was used in Bauer [16].

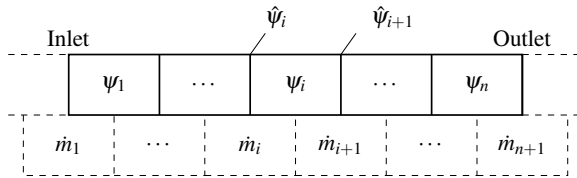


Figure 7: Staggered grid structure; thick = control volume grid, dashed = staggered grid

The mass and energy conservation equations become

$$A\Delta z \frac{d\bar{\rho}_i}{dt} = \dot{m}_i - \dot{m}_{i+1} \quad (21)$$

$$A\Delta z \frac{d}{dt} (\bar{\rho}_i \bar{h}_i - p_i) = \dot{H}_i - \dot{H}_{i+1} + \dot{Q}_i \quad (22)$$

where the enthalpy flow $\dot{H}_i = \dot{m}_i \hat{h}_i$ and heat flow $\dot{Q}_i = P\Delta z q''_{w,i} = P\Delta z h_{tc,i}(T_{w,i} - T_i)$ have been used, and Newton's law of cooling is applied with the well known heat transfer coefficient h_{tc} .

For convection dominated flows the upwind difference scheme is recommended to approximate thermodynamic quantities onto the staggered grid, because central difference scheme may lead to non-physical solutions. The 1st order upwind scheme is obtained by taking the control volume face value (staggered grid center) to be equal to the nearest upstream control volume center, thus

$$\hat{\psi}_i \approx \delta_i \psi_i + (1 - \delta_i) \psi_{i-1} \quad i = 1..n+1 \quad (23)$$

where δ_i is the indicator function denoting the direction of the mass flow

$$\delta_i = \begin{cases} 0 & \dot{m} \geq 0 \\ 1 & \dot{m} < 0 \end{cases} \quad (24)$$

The momentum equation becomes

$$\Delta z \frac{d\dot{m}_i}{dt} = \Delta \dot{I}_i - A(p_i - p_{i-1}) - F_{w,i} A \Delta z - \hat{\rho}_i g A \Delta z \sin \theta \quad (25)$$

where the momentum flow $\dot{I}_i = \dot{m}_i^2 / (\hat{\rho}_i' A)$ has been used and the difference in momentum flow, $\Delta \dot{I}_i$, is approximated according to the 2nd order central difference scheme as

$$\Delta \dot{I}_i \approx \frac{(\dot{I}_{i-1} - \dot{I}_i) + (\dot{I}_i - \dot{I}_{i+1})}{2} = \frac{d\dot{I}_{i-1} + d\dot{I}_i}{2} \quad (26)$$

where $d\dot{I}$ is the momentum flow difference between the staggered grid cells. The use of the central difference scheme serves to avoid discontinuities in the momentum equation.

Boundary models are used to compute the boundary conditions \dot{H} , \dot{I} , $d\dot{I}$, $\hat{\psi}$. The change of momentum flow $d\dot{I}$ at the inlet or outlet is simply set to zero, whereas the other variables are computed from the thermodynamic state and the mass flow rate.

Correlations for the frictional force, F_w , the heat transfer coefficient, h_{tc} , and the void fraction, α , must be supplied to close the system of equations.

A.2 Tube wall

The tube wall is discretized according to the Resistance Capacitance Method [19]. The method essentially uses the thermal resistances to describe the heat flows across the tube wall boundaries. The tube wall is assumed to have rotational symmetry, i.e. $T = T(r, z)$, and thus the energy equation for each discrete cell becomes

$$M_{c,p} \frac{dT}{dt} = \dot{Q}_W + \dot{Q}_E + \dot{Q}_S + \dot{Q}_N \quad (27)$$

where $\dot{Q}_S = -P\Delta z q''_w$ from equation 10. The entering and leaving heat flows are depicted on figure 8.

By definition, the heat flows are computed as $\dot{Q} = \Delta T / R$, where the thermal resistances in the radial and axial directions to the midpoint of the wall cell are

$$R_{ax} = 0.5 \frac{\Delta z}{kA} \quad (28)$$

$$R_{rad} = 0.5 \frac{\ln \frac{D/2}{d/2}}{2\pi k \Delta z} \quad (29)$$

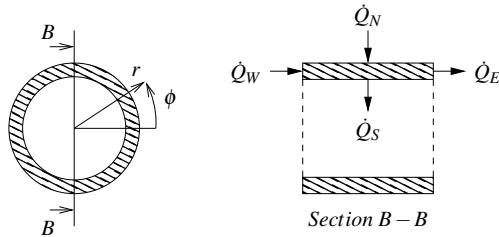


Figure 8: Heat flows to and from the tube wall

The boundary condition at the inlet and outlet of the pipe wall is simply no heat flow in the axial direction. Since we only use one cell per tube in this study the axial heat conduction is essentially neglected.

A.3 Airflow

The airflow is assumed to be incompressible and can not accumulate mass or energy. With these assumptions, the mass and energy conservation equation for each air cell become

$$\dot{m}_{in} - \dot{m}_{out} = 0 \quad (30)$$

$$(\dot{m}c_p T)_{in} - (\dot{m}c_p T)_{out} + \dot{Q}_N = 0 \quad (31)$$

The effectiveness-NTU method is applied to describe the variation in air temperature, i.e. the single stream heat exchanger configuration where the surface temperature of each cell is uniform. It describes the actual heat flow by the effectiveness, ϵ , of the highest possible heat transfer, i.e.

$$\dot{Q}_N = \epsilon C_{min}(-\Delta T_{max}) \quad (32)$$

where C_{min} is the minimum capacitance flow and ΔT_{max} is the maximum temperature difference. Correlations for the heat transfer coefficient and the fin efficiency must be applied to compute the Number of Transfer Units and thus the effectiveness.

A.4 Smooth functions

A first order continuous function is applied at the phase transitions ($0 \leq x < 0.05$ and $0.95 < x \leq 1$). The function ensures a smooth transition from two-phase to single phase in heat transfer and frictional pressure drop correlations. If the transitions are discontinuous, the equation solver might be slow or even fail to converge. The first order continuous function is described in Richter [10]. The used correlations are shown in table 2.

A.5 Heat exchanger architecture

Components of the refrigerant (both control volume grid cell and staggered grid cell), the wall and the air have been made in Dymola, and essentially arrays of these components are put together to form the evaporator in cross flow operation, as shown on figure 9.

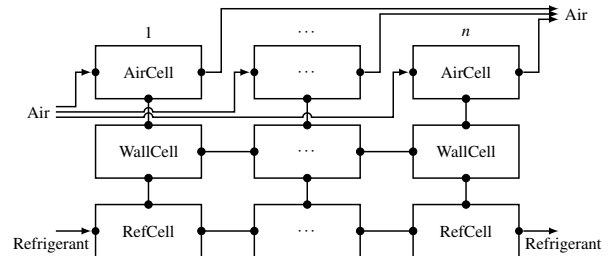


Figure 9: Heat exchanger architecture; cross flow.

Following this implementation, we did not use already made components from the Modelica standard library. We chose this to learn every step of the implementation in Modelica and to be able to quickly apply changes to the model formulation and correlations if necessary. Furthermore, we did not use the Modelica stream prefix. Since the compressor runs at constant speed, we did not observe flow reversal during the flow pulsations.

The circuitry modeling is a bit more complex than shown on figure 9, however, its construction is simply a matter of running through many for loops to connect the airflow paths and the refrigerant bends (assumed adiabatic) with correct radius. Note that the refrigerant flow is discretized fully from inlet to outlet through the bends such that the bends also contains a volume grid cell and a staggered grid cell. More information on the circuitry implementation is given in Kærn [3].

References

- [1] W. V. Payne, P. A. Domanski, Potential benefits of smart refrigerant distributors, Final report No. ARTI-21CR/610-20050-01, Air-Conditioning and Refrigeration Technology Institute, Arlington, VA, USA (2003).
- [2] J.-H. Kim, J. E. Braun, E. A. Groll, Evaluation of a hybrid method for refrigerant flow balancing in multi-circuit evaporators, International Journal of Refrigeration 32 (2009) 1283 – 1292.
- [3] M. R. Kærn, Analysis of flow maldistribution in fin-and-tube evaporators for residential air-conditioning systems, Ph.D. thesis, Technical

- University of Denmark, Department of Mechanical Engineering, Kgs. Lyngby, Denmark (2011).
- [4] P. A. Domanski, D. Yashar, Application of an evolution program for refrigerant circuitry optimization, in: ACRECONF "Challenges To Sustainability", New Delhi, India, 2007.
- [5] T. Funder-Kristensen, H. Nicolaisen, J. Holst, M. H. Rasmussen, J. H. Nissen, Refrigeration system, US Patent, Pub. No.: US 2009/0217687 A1 (2009).
- [6] G. Mader, C. Thybo, An electronic expansion valve with automatic refrigerant distribution control, in: Deutsche Kälte-Klima-Tagung, Magdeburg, Germany, 2010.
- [7] R. W. Fox, A. T. McDonald, P. J. Pritchard, Introduction to fluid mechanics, Wiley, New York, 2004.
- [8] C. Park, H. Cho, Y. Lee, Y. Kim, Mass flow characteristics and empirical modeling of R22 and R410A flowing through electronic expansion valves, International Journal of Refrigeration 30 (8) (2007) 1401–1407.
- [9] L. Chen, J. Liu, J. Chen, Z. Chen, A new model of mass flow characteristics in electronic expansion valves considering metastability, International Journal of Thermal Sciences 48 (6) (2009) 1235 – 1242.
- [10] C. C. Richter, Proposal of new object-oriented equation-based model libraries for thermodynamic systems, Ph.D. thesis, Technische Universität Carolo-Wilhelmina zu Braunschweig, Fakultät für Maschinenbau (2008).
- [11] M. R. Kærn, B. Elmegaard, L. F. S. Larsen, Experimental comparison of the dynamic evaporator response using homogeneous and slip flow modelling, in: 8th International Modelica Conference, Dresden, Germany, 2011.
- [12] Dynasim AB, Research Park Ideon SE-223 70, Lund, Sweden, Dynamic Modeling Laboratory, Dymola User's Manual, version 7.4 (2010).
- [13] J. Eborn, H. Tummescheit, K. Prölss, Airconditioning - a modelica library for dynamic simulation of ac systems, in: 4th International Modelica Conference, Hamburg, Germany, 2005, pp. 185 – 192.
- [14] M. J. Skovrup, Thermodynamic and thermophysical properties of refrigerants, Department of Energy Engineering, Technical University of Denmark, Nils Koppels Allé, Building 402, DK-2800 Lyngby, Denmark (2009).
- [15] W.-J. Zhang, C.-L. Zhang, A generalized moving-boundary model for transient simulation of dry-expansion evaporators under larger disturbances, International Journal of Refrigeration 29 (2006) 1119 – 1127.
- [16] O. Bauer, Modelling of two-phase flows with modelica, Master's thesis, Lund University, Department of Automatic Control (1999).
- [17] S. M. Ghiaasiaan, Two-phase flow: Boiling and Condensation in Conventional and Miniature Systems, 1st Edition, Cambridge University Press, 2008.
- [18] S. V. Patankar, Numerical heat transfer and fluid flow, Taylor & Francis, 1980.
- [19] A. F. Mills, Heat Transfer, 2nd Edition, Prentice Hall, 1999.

Validation and Application of the Room Model of the Modelica *Buildings* Library

Thierry Stephane Nouidui, Kaustubh Phalak, Wangda Zuo, Michael Wetter
Simulation Research Group, Building Technology and Urban Systems Department
Environmental Energy Technologies Division, Lawrence Berkeley National Laboratory
One Cyclotron Road, 94720, Berkeley, CA
TSNouidui@lbl.gov

Abstract

The Modelica *Buildings* library contains a package with a model for a thermal zone that computes heat transfer through the building envelope and within a room. It considers various heat transfer phenomena of a room, including conduction, convection, short-wave and long-wave radiation. The first part of this paper describes the physical phenomena considered in the room model. The second part validates the room model by using a standard test suite provided by the American Society of Heating, Refrigerating and Air-Conditioning Engineers (ASHRAE). The third part focuses on an application where the room model is used for simulation-based controls of a window shading device to reduce building energy consumption.

Keywords: Buildings library; ANSI/ASHRAE Standard 140; Simulation-Based Controls

1 Introduction

To support the design and operation of low energy buildings, the Lawrence Berkeley National Laboratory (LBNL) has been developing a free and open source Modelica *Buildings* library for building energy and control systems [1]. Version 1.1 Build1 of the library contains about 200 component models for building energy and control systems. These component models can be used for (1) rapid prototyping of innovative building systems, (2) design of building energy systems, (3) performance analysis of existing building systems, (4) development, specification and optimization of building control sequences, and (5) model-based operation for controls, fault detection and diagnostics.

Recently, we implemented window and room models into the *Buildings* library to extend its capability to whole building energy simulation [2]. However, the models were not systematically validated against reference data in [2]. In [3], we presented the valida-

tion of the window model which is an important part of the room model. This paper is to validate the room model and to show an application where the model is used as part of a controls framework of a window shading device of a building. After the introduction, we will briefly describe the physics and implementation of the room model. Then we will validate the room model using a subset of ANSI/ASHRAE Standard 140 [4], which is a standard test suite for evaluating building energy simulation tools. After validating the room model, we will describe an application where the room model is part of a simulation-based controls framework used to control a window shading device of a test cell for reducing building energy consumption.

2 Room model

The room model of the *Buildings* library simulates heat transport processes within rooms and through the building envelope. This model can be used for the modeling of rooms with unlimited number of opaque and transparent surfaces or entire buildings. The room model takes into account the following physical processes:

- (1) Transient or steady-state heat conduction through opaque surfaces, such as walls.
- (2) Heat transfer through glazing systems including solar radiation, infrared radiation from ambient environment, heat conduction and heat convection.
- (3) Convective heat transfer between the room (inside) air and room-facing surfaces using either a constant heat transfer coefficient or a temperature-dependent heat transfer coefficient.
- (4) Convective heat transfer between the outside air and outside-facing surfaces using either a constant heat transfer coefficient or a variable heat transfer coefficient as a function of wind-speed, wind-direction and temperature.
- (5) Solar and infrared heat transfer between the room enclosing surfaces.

(6) Temperature, pressure and species balance equations inside the room volume.

Note that the current room model assumes that the air in the room is well-mixed so that a single volume is used to represent the room air. More details of the room model are available in [2].

3 Validation of the room model

This section focuses on validation of the room model using different cases of ANSI/ASHRAE Standard 140 [4]. The Standard 140 is widely used in the building simulation community for testing the accuracy of building simulation models. Due to the complexity and high cost, it is difficult to precisely measure the energy performance of a building for a year. As an alternative approach, Standard 140 documents the simulated annual energy performance of a thermal zone using different building energy simulation tools. The simulation results of the tools are not the same since they use different assumptions, physical models and implementations. However, the variation of the simulation results is usually in a reasonable range. In this paper, we present validation cases of a low and high mass building using cases 600, 610, 620, 630, 600FF, 900, and 900FF.

Model configuration

For the validation, the following model configurations have been used:

- Room-side convective heat transfer coefficients are a function of the difference between air and surface temperature.
- Outside convective heat transfer coefficients are a function of the difference between air and surface temperature, and a function of wind speed.
- The long-wave radiative heat transfer has not been linearized.
- The medium model *Buildings.Media.GasesConstantDensity.SimpleAir* has been used.

For more details, all cases are available in the *Buildings* library version 1.2

3.1 Case 600: Low mass building without shading (South facing windows)

Case 600 is a low mass rectangular zone ($6\text{m} \times 8\text{m} \times 2.7\text{m}$) without interior partition and with two windows ($3\text{m} \times 2\text{m}$ each) on the south wall (Figure 1). Construction material properties and other details are provided in [4]. For the validation, we simulated the zone for a year with weather data provided in [4].

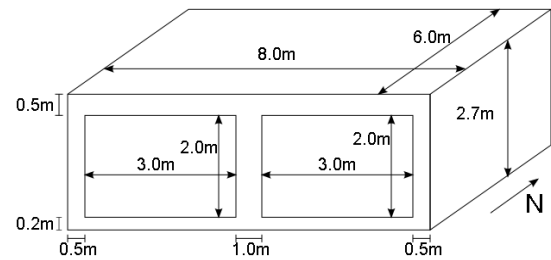


Figure 1 Case 600: Low mass rectangular zone

Figure 2 compares the annual heating and cooling loads calculated by the room model of the *Buildings* library with results of other energy simulation tools provided in [4]. The results of the room model, labeled as *Buildings Lib.*, are comparable with other energy simulation tools. The heating (5.44 MWh) and cooling (6.97 MWh) loads are within the range specified in [4].

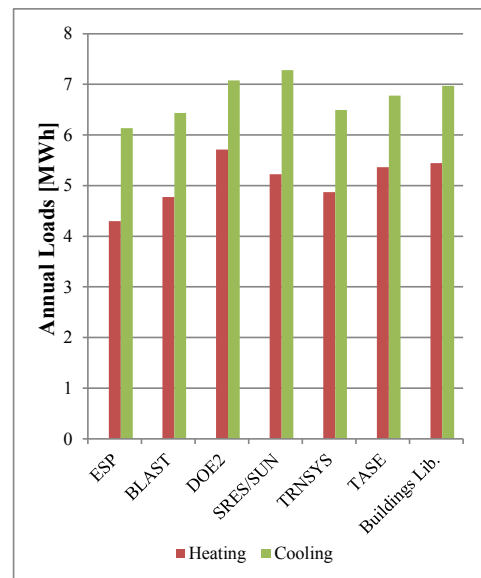


Figure 2 Case 600: Comparison of annual heating and cooling loads

We also compared the predicted peak heating load (Table 1) and peak cooling load (Table 2) and their time of occurrence. Again, the results of the *Buildings* library are in close agreement with simulation results of other tools. The difference observed in date of peak cooling load can be caused by different modeling assumptions in the simulation tools. The peak heating (4.23 kW) and cooling (6.82 kW) loads predicted by the *Buildings* library are within the minimum and maximum range specified in [4].

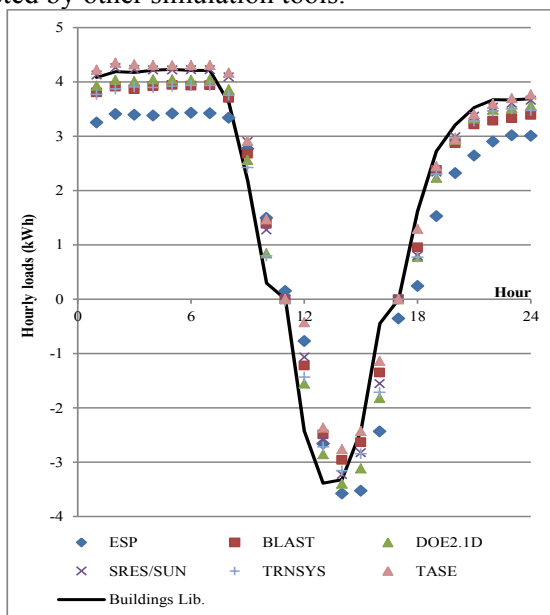
Table 1 Case 600: Annual hourly integrated peak heating loads

Code Name	kW	Date	Hour
ESP	3.437	4-Jan	5
BLAST	3.940	4-Jan	5
DOE2	4.045	4-Jan	5
SRES/SUN	4.258	4-Jan	2
TRNSYS	3.931	4-Jan	6
TASE	4.354	4-Jan	2
<i>Buildings Lib.</i>	4.229	4-Jan	5

Table 2 Case 600: Annual hourly integrated peak cooling loads

Code Name	kW	Date	Hour
ESP	6.194	17-Oct	13
BLAST	5.965	16-Oct	14
DOE2	6.656	16-Oct	13
SRES/SUN	6.827	16-Oct	14
TRNSYS	6.486	16-Oct	14
TASE	6.812	17-Oct	14
<i>Buildings Lib.</i>	6.821	17-Oct	13

Figure 3 shows hourly load profiles on the day of peak heating load (Jan 4th). In the load profiles, heating and cooling loads are represented with positive and negative values respectively. The *Buildings* library predicted that there was cooling load from about 11 a.m. to 5 p.m. and heating load for the rest of the day. This profile is similar to the profiles predicted by other simulation tools.


Figure 3 Case 600: Comparison of hourly heating and cooling load profiles for Jan 4th

3.2 Case 610: Low mass building with shading (overhang)

The case 610 is an extension of Case 600 in which a horizontal overhang is added to provide shading for

the south facing windows. The overhang is 1m deep, located at 0.5m above the windows and extends from east to west facing walls as shown in Figure 4. This case tests the ability of a simulation tool to treat shading of a south exposed window.

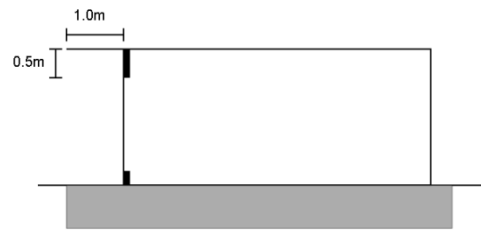

Figure 4 Case 610: Low mass building with overhang on south facing windows

Figure 5 compares the annual heating and cooling loads calculated by the *Buildings* Library with other simulation tools. The heating (5.47 MWh) and cooling (5.39 MWh) loads predicted by the *Buildings* library are within minimum and maximum range specified in [4]. As expected, adding shading device reduced the total cooling load. Compared to Case 600, the reduction in cooling load varied from 19% to 36% for different energy simulation tools. The *Buildings* library predicted a reduction of 23%. With less solar gain, all the programs also predicted increased (0.5% to 2%) heating load. The *Buildings* library predicted a minor increase of 0.6%.

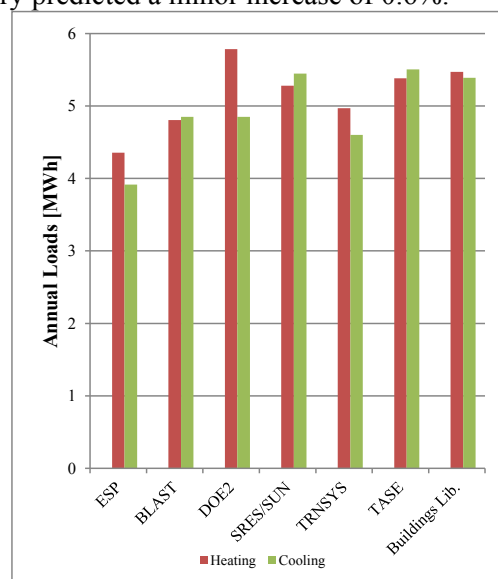

Figure 5 Case 610: Comparison of annual heating and cooling loads

Table 3 and Table 4 compare the predicted peak heating and cooling load and time of occurrence during the year. All simulation tools predicted almost similar time for the occurrence of the peak heating load. For peak cooling loads, two simulation tools predicted significantly different dates than the rest of the simulation tools. The room model predicted the same date as the majority of the tools. The *Buildings*

library calculated a peak heating load of 4.23 kW which is within the range of reference data. However, it slightly over-predicted the peak cooling load (6.38 kW) which is about 0.15% higher than the maximum value (6.37 kW) of the reference data.

Table 3 Case 610: Annual hourly integrated peak heating loads

Code Name	kW	Date	Hour
ESP	3.437	4-Jan	5
BLAST	3.941	4-Jan	5
DOE2	4.034	4-Jan	5
SRES/SUN	4.258	4-Jan	2
TRNSYS	3.922	4-Jan	6
TASE	4.354	4-Jan	2
<i>Buildings Lib.</i>	4.228	4-Jan	5

Table 4 Case 610: Annual hourly integrated peak cooling loads

Code Name	kW	Date	Hour
ESP	5.669	25-Nov	13
BLAST	5.824	25-Nov	14
DOE2	6.064	13-Jan	14
SRES/SUN	6.371	25-Nov	14
TRNSYS	5.675	25-Nov	14
TASE	6.146	17-Oct	14
<i>Buildings Lib.</i>	6.380	25-Nov	13

3.3 Case 620: Low mass building without shading (East-West facing windows)

The case 620 is same as Case 600 except that windows are oriented towards east and west as shown in Figure 6.

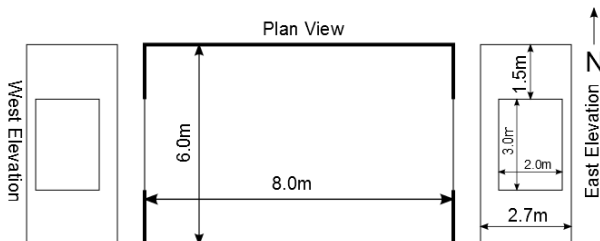


Figure 6 Case 620: East and West facing windows

Figure 7 compares annual heating and cooling loads computed by *Buildings Library* with other simulation tools. The results of room model (heating load: 5.61 MWh and cooling load: 4.31 MWh) are comparable with other simulation tools and are within the range specified in [4]. In contrast to Case 600 here heating load is higher than cooling as the room receives solar radiation during morning and evening when intensity of solar irradiation on the window surface is low, and during midday when the azimuth angle with respect to the window surface is high and the normal component of irradiation is low. Also both windows are never simultaneously exposed to the sun.

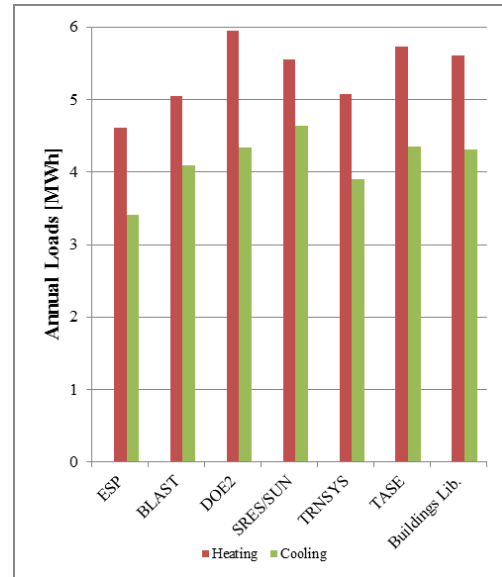


Figure 7 Case 620: Comparison of annual heating and cooling loads

Peak heating and cooling load with their time of occurrence is compared in Table 5 and Table 6. The results are comparable and are within range specified in [4]. Compared to Case 600 and Case 610 there is no significant change in peak heating load but peak cooling has reduced. This reduction is due to low solar heat gain as discussed earlier.

Table 5 Case 620: Annual hourly integrated peak heating loads

Code Name	kW	Date	Hour
ESP	3.591	4-Jan	6
BLAST	3.941	4-Jan	5
DOE2	4.046	4-Jan	5
SRES/SUN	4.277	4-Jan	2
TRNSYS	3.922	4-Jan	6
TASE	4.379	4-Jan	2
<i>Buildings Lib.</i>	4.230	4-Jan	5

Table 6 Case 620: Annual hourly integrated peak cooling loads

Code Name	kW	Date	Hour
ESP	3.634	26-Jul	16
BLAST	4.075	26-Jul	17
DOE2	4.430	26-Jul	17
SRES/SUN	4.593	26-Jul	17
TRNSYS	4.275	26-Jul	17
TASE	5.096	26-Jul	16
<i>Buildings Lib.</i>	4.295	26-Jul	16

3.4 Case 630: Low mass building with shading (overhang and window side fins)

Case 630 is an extension of case 620 in which an overhang and side fins are added on both east and west facing windows. The overhang is 1m deep, 3m wide and located 0.5m above the windows. The side

fins are 1m deep, along the vertical edges of the windows and extend from roof to ground level. This case tests the ability of the simulation tool to treat shading of east and west exposed windows with side fins and overhang combined.

As the east and west side windows are covered with overhang and side fins, the room receives little direct solar heat gain. This results in higher heating loads and lower cooling load. Results obtained from the *Buildings* library (heating load: 5.88 MWh, cooling load: 3.35 MWh) are comparable and within range of results from other simulation tools (Figure 8).

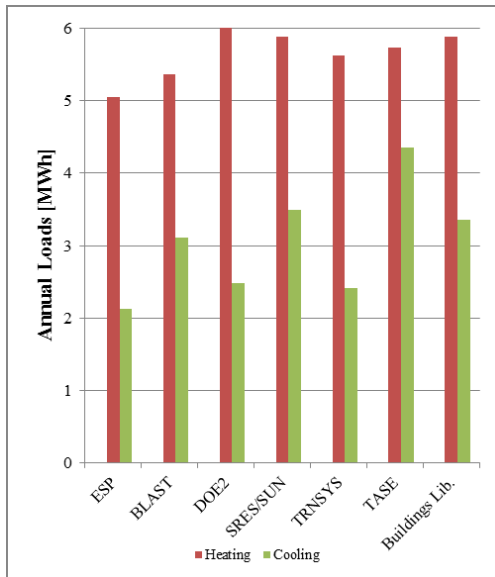


Figure 8 Case 630: Comparison of annual heating and cooling loads

Even though there is not much change in peak heating load compared to earlier cases, the peak cooling load has dropped significantly. In this case, both peak heating-cooling load and time of occurrence calculated by the *Buildings* library are within range and comparable with results from other tools as shown in Table 7 and Table 8.

Table 7 Case 630: Annual hourly integrated peak heating loads

Code Name	kW	Date	Hour
ESP	3.592	4-Jan	7
BLAST	3.941	4-Jan	5
DOE2	4.025	4-Jan	5
SRES/SUN	4.280	4-Jan	2
TRNSYS	3.922	4-Jan	6
TASE	N.A.	N.A.	N.A.
<i>Buildings Lib.</i>	4.230	4-Jan	5

Table 8 Case 630: Annual hourly integrated peak cooling loads

Code Name	kW	Date	Hour
ESP	3.072	26-Jul	16
BLAST	3.704	26-Jul	17
DOE2	3.588	26-Jul	17
SRES/SUN	4.116	26-Jul	17
TRNSYS	3.608	26-Jul	17
TASE	N.A.	N.A.	N.A.
<i>Buildings Lib.</i>	3.866	26-Jul	17

Low mass basic sensitivity tests

Sensitivity of each program for addition of overhang, side fins and change in window orientation is tested in [4] using differences in the results. The variation in annual and peak heating-cooling loads can be observed in Table 9 and Table 10 for Case 600 and Case 610. Results for *Buildings* library are within the range specified in [4].

Table 9 Difference in Case 600 and 610 results: Annual loads

Code Name	Heating [MWh]	Cooling [MWh]
ESP	0.059	-2.222
BLAST	0.033	-1.582
DOE2	0.077	-2.227
SRES/SUN	0.054	-1.830
TRNSYS	0.098	-1.891
TASE	0.021	-1.272
<i>Buildings Lib.</i>	0.029	-1.581

Table 10 Difference in Case 600 and 610 results: Peak loads

Code Name	Heating [kW]	Cooling [kW]
ESP	0.000	-0.525
BLAST	0.001	-0.141
DOE2	-0.011	-0.592
SRES/SUN	0.000	-0.456
TRNSYS	-0.008	-0.811
TASE	0.000	-0.666
<i>Buildings Lib.</i>	-0.001	-0.441

Differences in results of case 620 and 600 represent effect of change in window orientation. The differences in results of the *Buildings* library (Table 11 and Table 12) for these cases are within the range specified in [4]. This indicates that the room model correctly models modification in window orientation.

Table 11 Difference in Case 600 and 620: Annual loads

Code Name	Heating [MWh]	Cooling [MWh]
ESP	0.317	-2.72
BLAST	0.276	-2.341
DOE2	0.235	-2.745
SRES/SUN	0.328	-2.645
TRNSYS	0.201	-2.591
TASE	0.366	-2.427
<i>Buildings Lib.</i>	0.169	-2.661

Table 12 Difference in Case 600 and 620 results: Peak loads

Code Name	Heating [kW]	Cooling [kW]
ESP	0.154	-2.560
BLAST	0.001	-1.890
DOE2	0.001	-2.226
SRES/SUN	0.019	-2.234
TRNSYS	-0.008	-2.211
TASE	0.025	-1.716
<i>Buildings Lib.</i>	0.001	-2.526

As described earlier, in Case 630 overhang and side fins are added to the east and west facing windows of Case 620. The differences in results of these cases verify the effect of these shading devices. Table 13 and Table 14 compare the results of the *Buildings* library with other simulation tools.

Table 13 Difference in Case 620 and 630: Annual loads

Code Name	Heating [MWh]	Cooling [MWh]
ESP	0.437	-1.288
BLAST	0.310	-0.984
DOE2	0.525	-1.845
SRES/SUN	0.329	-1.140
TRNSYS	0.551	-1.485
TASE	N.A.	N.A.
<i>Buildings Lib.</i>	0.266	-0.956

Table 14 Difference in Case 620 and 630 results: Peak loads

Code Name	Heating [kW]	Cooling [kW]
ESP	0.001	-0.562
BLAST	0.000	-0.371
DOE2	-0.021	-0.842
SRES/SUN	0.003	-0.477
TRNSYS	0.000	-0.667
TASE	N.A.	N.A.
<i>Buildings Lib.</i>	0.000	-0.429

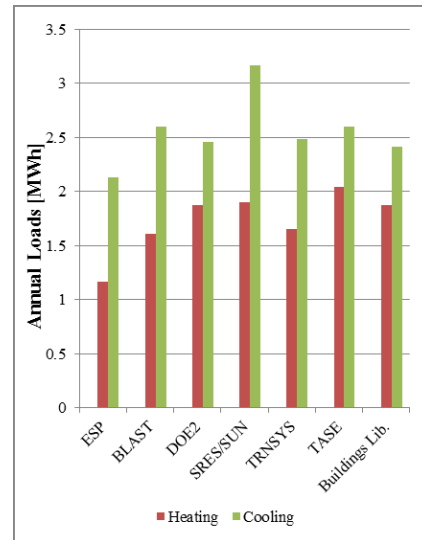
3.5 Case 600FF: Low mass building without temperature control

Case 600FF is based on case 600 except that there is no mechanical heating or cooling system. The room temperature is floating with the weather conditions. The *Buildings* library computed the highest room temperature (65.9°C) at 3 p.m. on October 17 and the lowest room temperature (-19.8°C) at 8 a.m. on January 4. These results are consistent with the ones computed by other simulation tools in Standard 140.

3.6 Case 900: High mass building with temperature control

Case 900 is a high mass building which uses the same building model as was used for Case 600 except that the wall and floor construction were changed to use heavier materials. This case is used to test the ability of a simulation tool to treat thermal

mass. As shown in Figure 9, the *Buildings* library predicted annual cooling and heating loads are in the range of Standard 140. The *Buildings* library also predicted the occurring hour for peak heating load (3.267 KW) at 7 a.m. on January 4 and peak cooling load (3.369 KW) at 2 a.m. on October 17. These values are also in the range of Standard 140.

**Figure 9** Case 900: Comparison of annual heating and cooling loads

3.7 Case 900FF: High mass building without temperature control

Case 900FF is the same as case 900 with the only difference that there is no mechanical heating or cooling system. The room temperature is floating. The *Buildings* library computed the highest room temperature (42.6°C) at 3 p.m. on September 2 and the lowest room temperature (-5.7°C) at 8 a.m. on January 4. These results are consistent with the ones predicted by other simulation tools in Standard 140.

4 Application

This section describes an application where the validated room model of the *Buildings* library was used in a simulation-based controls framework to control a window shading device of one test cell of the Advanced Windows Test Facility at LBNL (Figure 10). The windows facility is a test facility with three identical test cells which serve for testing and evaluation of controls strategies and façade systems. The dry bulb temperature in the corridor of the facility (Figure 11) is controlled to a constant value and the walls of the test cells are well insulated. This is to insure that all test cells experience the same load profiles. Each of the test cells has a floor area of about 14 m², a room volume of about 47m³ and a south facing window. The ovals in Figure 10 indicate the test cells that were used in this study. The win-

down shading device of the left test cell is controlled with the controls framework. The right test cell has a static interior blind and is used as our reference cell.

The room air temperature of the test cells is controlled to a fixed temperature. There are several sensors in the test cells which measure room air temperatures, exterior glass surface temperatures at the upper and lower window surfaces, plug loads, lighting loads, fan loads as well as transmitted solar irradiation at the upper and lower window surface. There are also several sensors located outdoors to measure external environmental conditions, such as solar irradiances, outdoor temperature, and wind speed (see Figure 12).



Figure 10 The Advanced Windows Test Facility at the Lawrence Berkeley National Laboratory

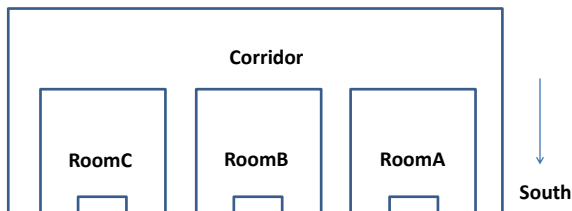


Figure 11 Schematic view of the Advanced Windows Test Facility at the Lawrence Berkeley National Laboratory

In this application, the room model of the *Buildings* library is used to model the test cell with the window and an exterior venetian blind. The window system installed in the test cell is a double pane window. The exterior venetian blind can be remotely controlled to be fully retracted or fully closed. It is also possible to control the slat angle positions of the blind.

In the following sections, we will describe the controls framework applied to control the blind of the window system. The objective of the framework is to control the blind to reduce heating and cooling loads of the test cell. The indoor dry-bulb temperature was controlled to a constant value of 24°C. To reduce the heating and cooling loads, an optimal blind position is calculated at discrete time steps using Modelica models of the *Buildings* library and a control algorithm. This position is then converted into a control

signal which is sent to real hardware to move the blind in the desired position.

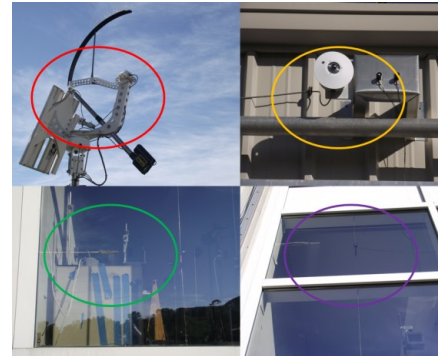


Figure 12 Instrumentation used at the test facility (Pyranometer (top left), pyrgeometer (top right), temperature sensors (bottom left), pyranometer (bottom right))

4.1 Overview of the Controls Framework

Figure 13 shows the schematic of the framework for one simulation time step. It involves the co-simulation between different simulation tools and the communication between hardware and software. The entire process is controlled by the Building Controls Virtual Test Bed (BCVTB) [5]. The BCVTB is an open source software environment developed by LBNL and based on the Ptolemy II software from UC Berkeley [6]. It allows expert users to couple different simulation programs for co-simulation, and to couple simulation programs with actual hardware [7].

In the controls framework, the BCVTB is the master that orchestrates the simulations and data exchange among simulators and hardware. It sets the start time, the stop time as well as the sampling time when blind position should be updated. It uses the *SystemCommand* actor [5] to call scripts which start different simulation programs. In our implementation, the simulation runs in real-time with a time step size of 5 minutes.

The simulation workflow can be divided into 8 steps. At the beginning of the simulation, the BCVTB gets the start and end time of the simulation, the test cell number, and the time step that are pre-defined by the users.

In step (1) of every time step, it uses a Python [8] script to send requests through the internet to get the current clock-time, weather data, as well as plug, fan and lighting loads which are measured in the test cell.

In step (2), it writes a weather file and a load file. The weather file contains measured weather data including diffuse solar irradiation on the horizontal surface, direct solar irradiation, the atmospheric infrared solar irradiation, outdoor dry-bulb tempera-

ture, and wind speed. The load file contains the sum of plug, fan and lighting loads.

In step (3), the BCVTB starts a Perl [9] script which invokes Radiance [10] to calculate the incoming solar irradiations and the solar radiation absorbed by different room surfaces for multiple blind positions. Radiance is a ray-tracing based daylighting simulation program. It is selected because it can compute light transmittance of complex fenestration systems with light-redirecting shades. Since the *Buildings* library does not support the modeling of venetian blinds, we use the capability of Radiance to compute the light redirection of the blinds, and to compute the solar irradiation distribution in the room. This was achieved by calculating incident and absorbed solar irradiation in Radiance for distinct blind positions and overwriting the solar irradiation distribution calculations done in the room model. In our configuration, we considered 11 positions. Because the simulations were fast compared to the sampling time, and only 11 control options need to be considered, we did an exhaustive search to determine the optimal control signal. The first position is with the blind fully retracted. The second to the 11th position are with the blind set at angles with degree of 40, 35, 30, 25, 20, 15, 10, 5, 0, and -5, respectively, where the last position is with the blind fully closed. The calculated irradiation data includes incident solar radiation on interior wall surfaces of the test cell and solar irradiation absorbed in glass layers and the shading layer of the window system. At the end of the calculation, the results are written to the files which will be used for step (4).

In step (4), the BCVTB starts a script, which simulates multiple instances of the Modelica room model using Dymola [11]. Each model represents the room with the blind set to a specific position. The model is parameterized using a weather file, load file

as well as incoming and absorbed solar irradiation pre-calculated by Radiance. Figure 14 shows a screenshot of the Modelica implementation of the test cell. This model consists of 7 parts: part 1 defines the heat sources which are read from the load file, part 2 is the PI controller for heating, part 3 is the PI controller for cooling, part 4 models the building envelope, part 5 represents the material properties of the building envelope, part 6 provides the weather data, and part 7 computes the infiltration in the test cell.

In step (5), the BCVTB calls a Python script to collect the Modelica simulation results for different blind positions and determines the optimal position which will lead to the least heating and cooling load. This position is then written in a file named "chosenposition.txt".

In step (6), the BCVTB calls a script which saves the state variables of the room model with the optimal blind position. These state variables will be used as initial conditions in the next time step. The capability of Modelica to easily reinitialize state variables, the transparency of making changes to models and the separation between process model, control implementation and numerical methods are important reasons why Modelica being well suited suitable for simulation-based controls operations.

In step (7), the BCVTB calls a script which reads the optimal blind position from the "chosenposition.txt" file, converts it into a controls signal, and sends it through the internet to the actuator to set the position of the blind.

Finally, in step (8), the BCVTB calls a script which requests the hardware to report the actuation position set. This is written in a log file. The BCVTB then pauses until the next time step is reached and restarts the process.

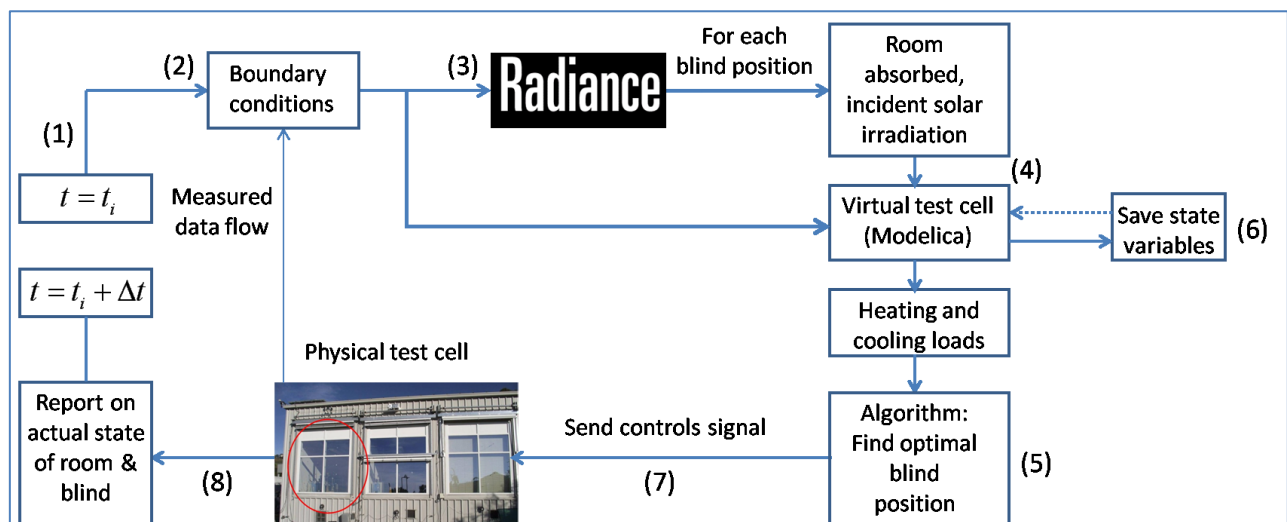


Figure 13 Simulation-based controls framework used to control one of the test cells of the test facility

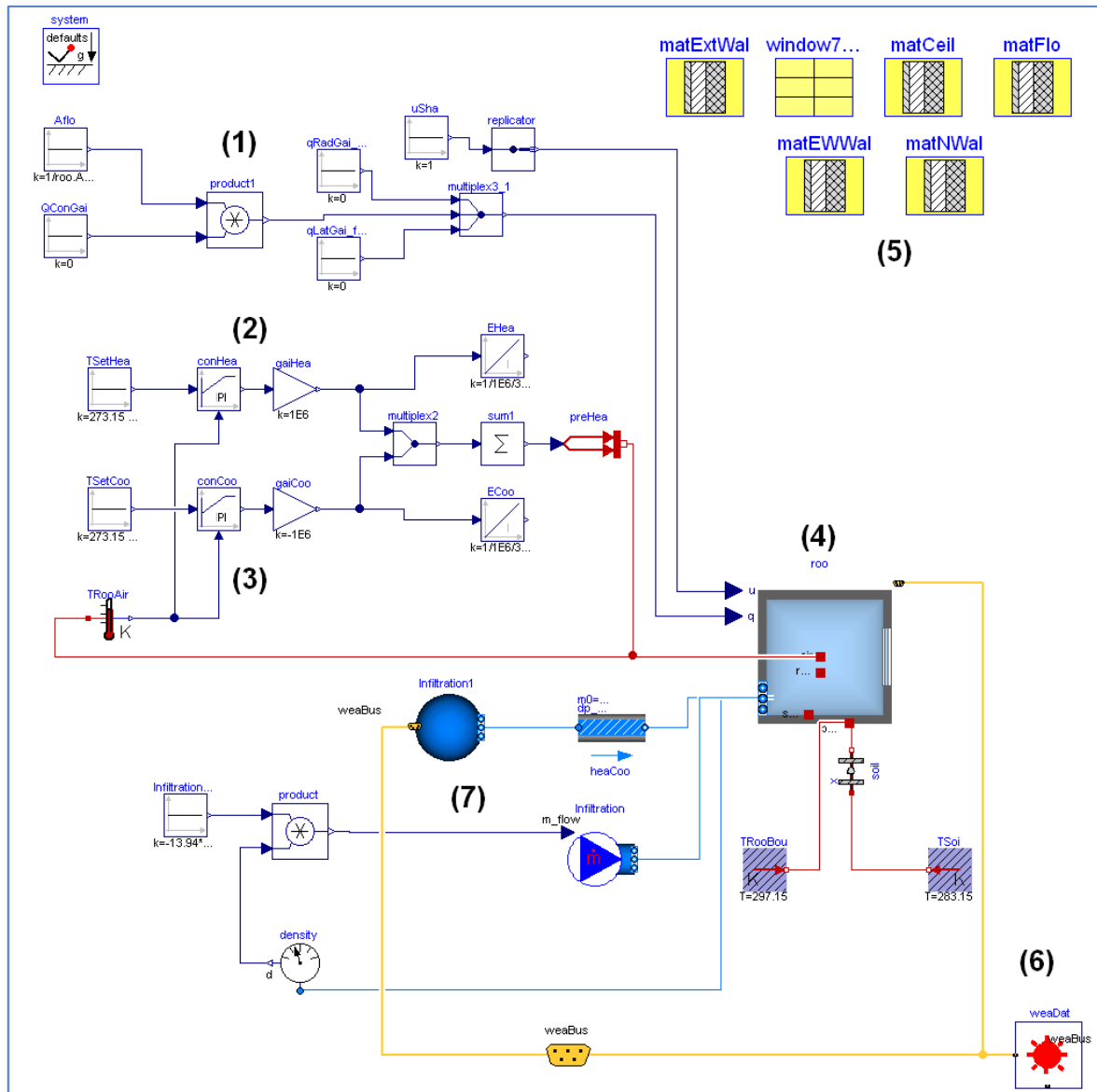


Figure 14 Modelica implementation of the test cell

4.2 Simulation results

In our preliminary work, we measured the heating and cooling loads of two test cells for a period of 9 days (from 04/13/2012 to 04/22/2012). One test cell used an interior static venetian blind set at 30 degree blocking angle (RoomA). This represents one common configuration for blinds which is generally set by users. The other test cell (RoomC) has an exterior blind controlled using the simulation-based controls framework.

As shown in Figure 16 the heating and cooling load of the test cell with controlled exterior venetian blind is much less than that with the interior static blind. The measurements show in the peak up to two and half times lower cooling load in the room with the controlled exterior venetian blind. Consequently,

one can save cooling energy by using the controlled exterior venetian blind.

Considering the test was only about one week and there were days with missing data, further investigations are needed to evaluate the performance of the algorithm over a longer period of time. Both exterior blind and controls can contribute to the energy saving in current study. To quantify the energy saving due to the controls, we will need to use exterior venetian blinds for both test cells. Nevertheless, the preliminary results show that our controls framework is functioning and the Modelica room model can meet the requirements of the application.

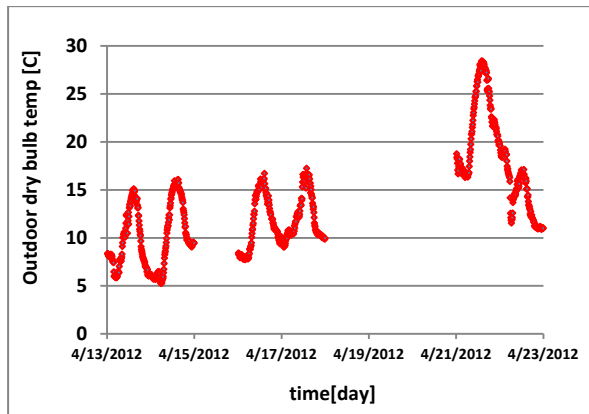


Figure 15 Measured outdoor dry bulb temperature

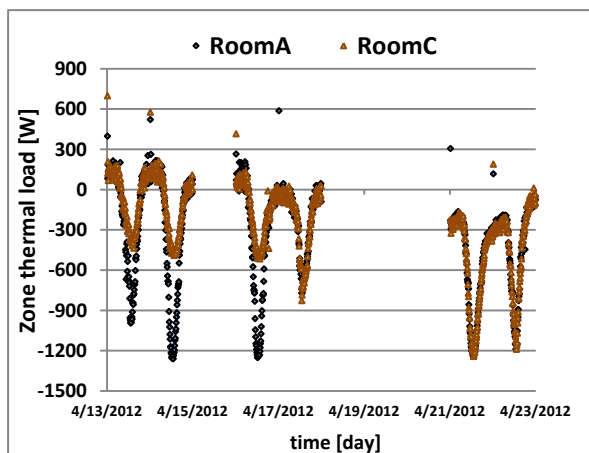


Figure 16 Comparisons between heating and cooling loads derived from measurements obtained in RoomA (static blind) and RoomC (controlled blind)

5 Conclusions

The validation results show that the room model of the Modelica *Buildings* library generates similar results for low and high mass buildings with and without shade compared to other energy simulation tools listed in ANSI/ASHRAE Standard 140. The application shows how the room model of the Modelica *Buildings* library can be used as part of a simulation-based controls framework of shading. This demonstrates that the room model of the Modelica *Buildings* library can be used not only for whole building simulations, but also as part of a framework for simulation-based controls operations.

Acknowledgments

This work was supported by the Assistant Secretary for Energy Efficiency and Renewable Energy, Building Technologies Program of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231 and by the California Energy Commission, Public Interest Energy Research Program, Buildings End Use Energy Efficiency Pro-

gram, award number 500-10-052. We would like to thank Andrew McNeil for his support in developing and integrating scripts for radiance calculations. We would like to thank Eleanor Lee for allowing us access to the Advanced Windows Test Facility.

References

- [1] M. Wetter, W. Zuo and T. S. Noudui, "Recent developments of the Modelica buildings library for building energy and control systems," in *Proceedings of the 8th International Modelica Conference. Dresden, Germany, March 2011*, 2011.
- [2] M. Wetter, W. Zuo and T. S. Noudui, "Modeling of Heat Transfer in Rooms in the Modelica "Buildings" Library," in *Proceedings of Building Simulation 2011*, Sydney, 2011.
- [3] T. S. Noudui, M. Wetter and W. Zuo, "Validation of the window model of the Modelica Buildings library," in *Proceedings of SimBuild2012*, Madison, 2012.
- [4] ANSI/ASHRAE, Standard Method of Test for the Evaluation of Building Energy Analysis Computer Programs (ANSI/ASHRAE Standard 140-2007), Atlanta: American Society of Heating, Refrigerating and Air-Conditioning Engineers, 2007.
- [5] M. Wetter, "Co-simulation of building energy and control systems with the Building Controls Virtual Test Bed," *Journal of Building Performance Simulation*, vol. 3, no. 4, 2011.
- [6] C. Brooks, E. A. Lee, X. Liu, S. Neuendorffer, Y. Zhao and E. A. L. X. L. Haiyang Zhengher, "Ptolemy II – Heterogeneous Concurrent Modeling and Design in Java," Berkeley, 2007.
- [7] T. S. Noudui, M. Wetter, Z. L. Li, X. Pang, P. Bhattacharya and P. Haves, "BACnet and Analog/Digital Interfaces of the Building Controls Virtual Test Bed," in *Proceedings of the 12th IBPSA Conference, p. 294--301. Sydney, Australia, November 2011*, 2011.
- [8] PYTHON. [Online]. Available: <http://www.python.org/>.
- [9] PERL. [Online]. Available: <http://www.perl.org/>.
- [10] RADIANCE. [Online]. Available: <http://radsite.lbl.gov/radiance/index.html>.
- [11] Dymola. [Online]. Available: <http://www.3ds.com>.

The Indoor Climate Library and its Application to Heat and Moisture Transfer in a Vehicle Cabin

Victor Norrefeldt¹, Daniel Andersson², Arnav Pathak¹, Hubertus Tummescheit²

1: Fraunhofer-Institute for Building Physics

Fraunhoferstr. 10, D-83626 Valley, Germany

2: Modelon AB

Ideon Science Park, Beta 6 building, Scheelevägen 17, S-22370 Lund, Sweden

victor.norrefeldt@ibp.fraunhofer.de, daniel.andersson@modelon.com,

arnav.pathak@ibp.fraunhofer.de, hubertus.tummescheit@modelon.com

Abstract

This paper presents the newly developed Indoor Climate Library. The library facilitates simulation of the coupled heat and moisture transfer through envelopes and the interaction of envelopes with the interior air. The computation of coupled heat and moisture transfer becomes more and more important for the development of electric vehicles. Due to the lack of waste heat from the combustion engine the heating of a vehicle cabin during winter time becomes a challenge. One way to reduce heat losses through the envelope is to add insulation. However, insulation bears the risk of water accumulation and its performance usually decreases with increased water content. The Indoor Climate Library helps the user to detect such problems early in the product development process and to find remedies.

Keywords: Heat and moisture transfer; Indoor air, Modelica Library

1 Introduction

To compute heat and moisture flow through building wall constructions, Nouidui [1] has built the Building Physics Library as research code. The authors have now rearranged and updated this code with the focus on user friendliness and increased applicability. Templates have been added allowing the quick setup of a model. Furthermore, the package structure has been rearranged to allow the user to easily navigate the library.

In Modelica different libraries are provided for the computation of building related problems. The

Buildings Library [2-5] contains thermal models for walls, windows, shading systems, HVAC components, controls, etc. Even components from the Modelica Standard library allow quick setup of thermal models of wall constructions. However, these libraries are limited to the thermal aspect of energy flows but neglect the moisture flow through constructions. Raised moisture levels adversely affect material properties. Risk of mold growth increases resulting in a harmful indoor environment [6]. Thermal conductivity of insulation materials usually increases with moisture content resulting in a degradation of insulating properties. Furthermore, the moisture transfer itself leads to a considerable enthalpy flow when evaporation or condensation occur, affecting wall temperatures considerably. The goal of the Indoor Climate Library is to provide a tool that predicts heat and moisture flows. Based on outside weather conditions the temperature and humidity profile in the enclosure layers and of the inner air are computed. The classical application field of the library is for buildings. However, recent developments of energy efficient heating systems for electrical vehicles show the need of using more insulation materials. Therefore, the library focuses on applications in the automotive and aviation sector as well.

2 Use of the Indoor Climate Library

The following section describes how to use the Indoor Climate Library.

Nomenclature	
A	Area [m ²]
A_w	Absorption coefficient [kg/(m ² ·h ^{0.5})]
c	Specific heat capacity of dry material [J/kg·K]
c_w	Specific heat capacity of water [J/kg·K]
d	Thickness [m]
D_w	Liquid transport coefficient [m ² /s]
D_{wr}	Liquid transport coefficient at redistribution [m ² /s]
D_{ws}	Liquid transport coefficient at suction [m ² /s]
f	Form factor [-]
H	Enthalpy [J/kg]
h_c	Convective heat transfer coefficient [W/m ² ·K]
\dot{m}	Mass flow rate [kg/(m ² ·s)]
p_{water}	Water vapour pressure [Pa]
\dot{q}	Heat flow rate [W/m ²]
T	Temperature [K]
w	Water content [kg/m ³]
w_f	Free water saturation [kg/m ³]
w_{max}	Maximum water content [kg/m ³]
β_c	Convective mass transfer coefficient [kg/(m ² ·Pa)]
δ	Water vapor permeability [m ² ·s]
ϵ	Emissivity [-]
λ	Thermal conductivity [W/m·K]
μ	Water vapor diffusion number [-]
ρ	Density [kg/m ³]
σ	Planck constant $5.67 \cdot 10^{-8}$ [W/(m ² ·K ⁴)]
Subscripts	
i, j	numeration indexes
l	liquid
v	vapor

2.1 Before modeling

Before modeling the user needs to answer the following questions:

- How many domains are needed
- How many walls are needed
- What materials are used
- How many windows are needed
- What window types are used
- To which domains do these walls and windows connect
- How many outside surfaces are needed
- How are the surfaces oriented
- Time and place
- Which weather data to use

When having found an answer to these questions, the user can build the whole model from predefined parameterized templates: The wall and window templates allow quick creation of models of different enclosures. The domain model contains a model of the air in a room that is connected to the walls and windows. Outside surfaces are the interface between wall templates and the environment. The environment provides the boundary conditions of the simulation.

2.2 Wall Template

The wall template consists of ten material layers. The default model for a material layer is the “None”-

model. This is a passive model that can be exchanged by the needed material layer models. To configure the wall model, the user selects the needed material from a drop-down list.

Figure 1 shows the parameter dialog of a material layer. The number of nodes, the layer thickness, the discretization scheme and initial conditions must be set. The default discretization scheme uses small nodes near material layer boundaries and larger nodes in the middle. By changing the status of a radio button the user can choose to define a custom discretization. Another radio button allows the user selecting to enter the initial water content or the initial relative humidity of the material. Furthermore, the initial temperature can be set.

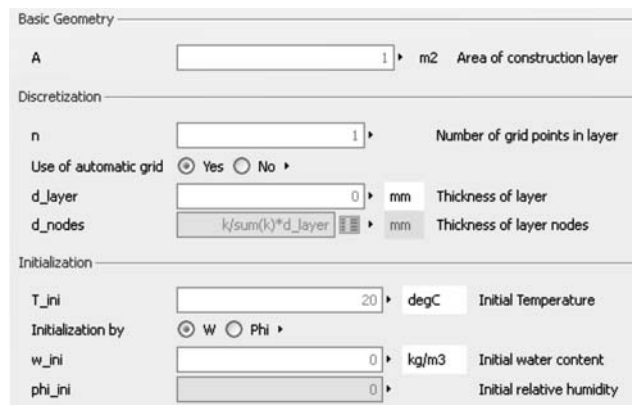


Figure 1: Parameterization of the material layer model

2.3 Window Template

The type of window is selected from a drop-down list. Models for one, two and three-pane windows are available. Heat transfer through conduction, convection, long-wave radiation, transmission and absorption of solar radiation are taken into account when computing pane temperatures. The transmitted solar radiation is propagated to the adjacent domain.

2.4 Domain template

A rectangular room is a simple example for a domain. It consists of an air volume and of six inside surfaces. The surface is considered as the infinitesimally narrow layer between the air volume and the wall. The wall side of the surface transports heat and moisture by conduction. The air side of the surface exchanges heat and moisture convectively with the adjacent air volume. A radiation node estimates the radiation between surfaces. View factors in the radiation node are computed from the connected surface's relative absorption weighted areas. For building applications this approach is sufficiently accurate [4]. If the user possesses more advanced view factors, a more detailed radiation model taking the real view factors into account, can be used. Windows are treated like any other wall in the domain model except that a further connection to a radiative source node is needed for transmitted solar radiation.

Figure 2 shows the parameterization of a domain model. The user gives the number of surfaces, their area, convective heat and moisture transfer coefficients or correlations and long-wave emissivities. If radiative or convective heat sources are contained in the domain their number must be given and corresponding models connected with the domain. For the air volume, the volume and initial pressure, temperature and relative humidity must be entered.

Type a Surfaces		
n_surfaces	1	Number of surfaces
A	ones(n_surfaces)	m2 Area of a-type surfaces
h_c	ones(n_surfaces)	W/(m2.K) Heat transfer coefficient for surfaces
beta_c	ones(n_surfaces)	kg/(s.m2.Pa) Moisture diffusion coefficient for surfaces
epsilon_lw	ones(n_surfaces)	Long-wave emissivity of surfaces
Heat Sources		
n_source_rad	0	Number of radiative sources in zone
n_source_cv	0	Number of convective sources in zone
Air Volume		
V	1	m3 Volume of air
p_ini	1.01325	bar Initial Air Pressure
T_ini_air	20	degC Initial Air Temperature
phi_ini_air	0.5	Initial Relative Humidity of air

Figure 2: Parameterization of the domain model

2.5 Outside Surfaces

The geometric parameters of an outside surface are area, slope and azimuth angle (Figure 3). The surface model has an outer instance of the environment model.

As for the inside surface, heat and moisture is transported by conduction on the wall side and by convection on the air side. Radiation and radiation parameters are split into long-wave and short-wave radiation. Long wave radiation is exchanged with surrounding earth and with the sky. Short wave radiation is provided by the sun. A distinction is made between direct and diffuse solar radiation. A geometrical model computes the impact angle of the sun to determine the direct solar radiation. This angle depends on the slope and azimuth of the surface, location and time. Diffuse radiation is independent of the surface orientation, e.g. light also enters through a north facing window during daytime.

Geometry		
A	1	m2 Surface area
Slope	90	deg
SurfaceAzimuthAngle	0	deg Surface Facing South is zero, West positive, East negative, -180° <= surface azimuth angle <= 180°
Radiation		
a_sw	0.9	Short-wave radiation absorptivity of surface
eps_lw_surf	0.9	Emissivity of surface

Figure 3: Parameterization of the outer surface model

2.6 Environment model

The environment model is used at the top-level of simulation models as an inner component. Information comes from weather files of test reference year data and is used in all models referring to outside weather conditions. The outputs from the component are the air temperature, humidity and pressure, wind speed and direction, intensity of direct and diffuse solar radiation, intensity of terrestrial and sky radiation and cloudiness.

The user selects a weather data file and the location of the building or cabin. This location is important for the geometrical sun model. Depending on the time format of the weather data file, a correction of the local standard time longitude needs to be entered. For GMT this correction is zero, for CET -15°. To assess the reflection of solar and sky radiation by the soil, the corresponding parameters need to be entered. Meaningful standard values are set as default. The user needs to provide the start time and date of the simulation and the start time and date of weather data. This is necessary to align the weather data, the sun position and the simulation time in the integrator.

Weather Data

WeatherRecord:

Location

Longitude: deg East: negative, West: positive

Latitude: deg North positive, South negative

LocalStandardTimeLongitude: deg East: negative, West: positive

Radiative Properties of Environment

rho_solar_env: Solar reflectance of environment

r_terr: Long-wave reflectance of soil

Start time of simulation

startMonth: Starting Month of simulation

startDay: d Starting Day of simulation

startHour: h Starting Hour of simulation

startMonthWeatherData: Starting Month of weather data

startDayWeatherData: d Starting Day of weather data

startHourWeatherData: h Starting Hour of weather data

Figure 4: Parameterization of the environment model

3 Technical background of the Indoor Climate Library

In this section, the principles of hygrothermal simulation are described.

3.1 Water storage function

The water storage function describes the relation between water content and relative humidity of a material. This function is usually non-linear and often increases more steeply at higher relative humidity. It needs to be determined experimentally. At 100% relative humidity free water saturation w_f is reached. An example of water storage functions is shown in Figure 5.

The porosity of a material indicates the maximal water content w_{max} . When all pores are filled with liquid water the material cannot be further penetrated by water. This maximal water content is above the free water saturation provided by the moisture storage function. In the range between the free water saturation and maximal water content the relative humidity remains equal to one, and is therefore independent of the water content. Up to the free water saturation, the material can be in an equilibrium state. Above no boundary condition exist that could maintain the reached water content [8].

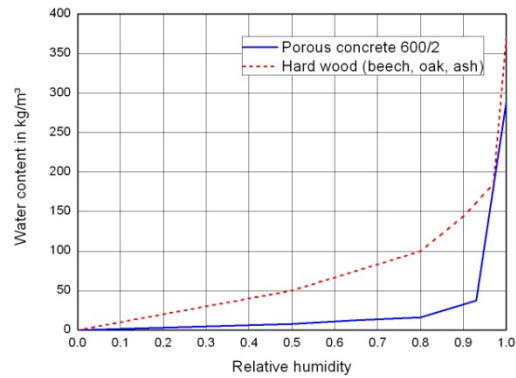


Figure 5: Examples of water storage functions [7]

3.2 Heat flow

The heat flow through a material node is obtained from the difference of the temperature T between nodes i and $i+1$, the conductivity λ_i and the length d_i of node i :

$$\dot{q}_i = \lambda_i \cdot \frac{T_i - T_{i+1}}{d_i} \tag{1}$$

For some materials thermal conductivity is constant, e.g. concrete: $\lambda=0.24$ W/(m·K). Other materials like mineral wool show an increase of thermal conductivity at higher water contents (Figure 6). The Indoor Climate Library uses a replaceable thermal conductivity model to match the type of material.

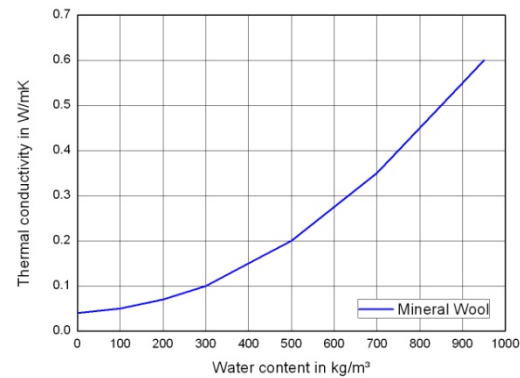


Figure 6: Example for increase of thermal conductivity with water content [7]

3.3 Water vapor diffusion

The driving potential for water vapor diffusion is the difference of the water vapor pressure p_{water} between nodes i and $i+1$.

$$\dot{m}_{v,i} = \delta_i \cdot \frac{p_{water,i} - p_{water,i+1}}{d_i} \tag{2}$$

The permeability δ_i of a material to water vapor is obtained from a function depending on the material node's temperature T_i and the water vapor diffusion number μ_i (equation (3)). This number is a property of the material; for stagnating air it is by definition one. Depending on the type of material it can be constant (e.g. porous concrete 600/2: $\mu=6.7$) or vary with relative humidity (Figure 7).

$$\delta_i = \frac{2 \cdot 10^{-7} \cdot T_i^{0,81}}{101300 \cdot \mu_i} \quad (3)$$

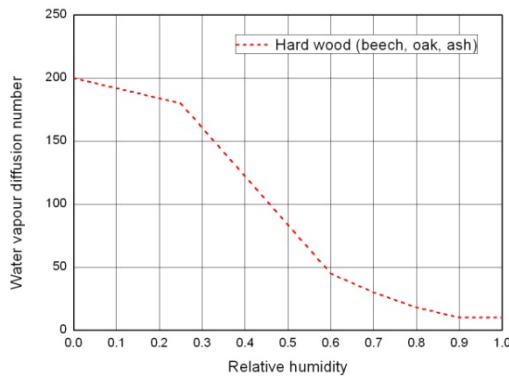


Figure 7: Example of relative humidity dependent water vapor diffusion coefficient [7]

3.4 Liquid water transport

Some materials are able to transport liquid water by capillary suction. Liquid transport is driven by the difference of water content w_i between material node i and $i+1$.

$$\dot{m}_{l,i} = D_{w,i} \frac{w_i - w_{i+1}}{d_i} \quad (4)$$

The liquid transport coefficient D_w depends on whether the material surface is wet due to rain or whether it is dry. On a wet surface suction occurs, if the surface is dry redistribution occurs. The redistribution factor D_{wr} can be estimated by a factor of 10 smaller than the suction factor D_{ws} [9].

Often, the water absorption coefficient A_w is given. For example, porous concrete has a A_w coefficient of $5.4 \text{ kg}/(\text{m}^2 \cdot \text{h}^{0,5})$. Künzel [8] suggests equation (5) to compute the liquid transport coefficient at suction from A_{ws} , the free water saturation and the actual water content:

$$D_{ws,i} = 3.8 \cdot \left(\frac{A_w}{w_f} \right)^2 \cdot 1000 \left(\frac{w_i}{w_f} - 1 \right) \quad (5)$$

3.5 Heat and Moisture Balance for a material layer node

The sum of entering and leaving heat and water flows yields the variation of temperature and water content of a material. It is admitted that water vapor enters node i with temperature T_{i-1} and condenses at temperature T_i . Similarly it evaporates and leaves at temperature T_i . Liquid water is admitted to enter with temperature T_{i-1} and to leave at temperature T_i . To describe this process the evaporation and liquid enthalpies H_v and H_l are introduced to the heat balance equation. The thermal inertia is the sum of the dry thermal inertia $\rho \cdot c$ (density, specific heat capacity) and the thermal inertia of water contained in the node $w_i \cdot c_w$ (c_w : specific heat capacity of water).

$$\begin{aligned} d_i \cdot [(\rho \cdot c + w_i \cdot c_w) \cdot \dot{T}_i + H_{l,i} \cdot \dot{w}_i] = \\ \dot{q}_{i-1} - \dot{q}_i \\ + \dot{m}_{v,i-1} \cdot (H_{v,i-1} - H_{l,i}) \\ - \dot{m}_{v,i} \cdot (H_{v,i} - H_{l,i}) \\ + \dot{m}_{l,i-1} \cdot (H_{l,i-1} - H_{l,i}) \end{aligned} \quad (6)$$

The variation of the water content w_i in node i is obtained from the sum of entering and leaving mass flows.

$$\begin{aligned} d_i \cdot \dot{w}_i = \\ \dot{m}_{v,i-1} - \dot{m}_{v,i} + \dot{m}_{l,i-1} - \dot{m}_{l,i} \end{aligned} \quad (7)$$

3.6 Surfaces

A surface exchanges heat and moisture between air and the adjacent material layer. The heat exchange takes into account convection due to the temperature difference between wall and air (h_c : convective heat transfer coefficient) and the enthalpy flow of the exchanged water vapor (equation (8)).

The moisture flow is determined by the convective moisture transfer coefficient β_c and the water vapor pressure difference (equation (9))

$$\begin{aligned} \dot{q}_{cv} = h_c \cdot (T_{air} - T_{wall}) \\ + \dot{m}_v \cdot (H_{v,air} - H_{v,wall}) \end{aligned} \quad (8)$$

$$\dot{m}_v = \beta_c \cdot (p_{water,air} - p_{water,wall}) \quad (9)$$

Inner surfaces exchange heat by radiation. An approximated form factor of a surface is obtained by equation (10), where ϵ is the long-wave emissivity of the surface and A its area. The radiation between surfaces is estimated in a radiation node model that distributes radiation between surfaces. Radiative

sources are distributed on all surfaces contained in the domain (equation (11))

$$f_i = \frac{\epsilon_i \cdot A_i}{\sum_{domain} \epsilon_j \cdot A_j} \quad (10)$$

$$\begin{aligned} \dot{q}_{rad,lw,i} &= \sigma \cdot \epsilon_i \cdot \sum_{j \in domain} f_j \cdot (T_i^4 - T_j^4) \\ &+ \sum_{sources} f_i \cdot \dot{q}_{rad,source} \end{aligned} \quad (11)$$

For outer surfaces, the long wave terrestrial radiation, the long wave atmospheric radiation and the short wave solar radiation are taken into account.

4 Application example

An insulated car cabin (Figure 8) is considered as application example. Four passengers are supposed to travel one hour in the morning and one hour in the evening from Monday to Friday in the region of Holzkirchen, Germany, during January 2011. During weekend the car is not used. Passengers emit heat and moisture according to sedentary work.

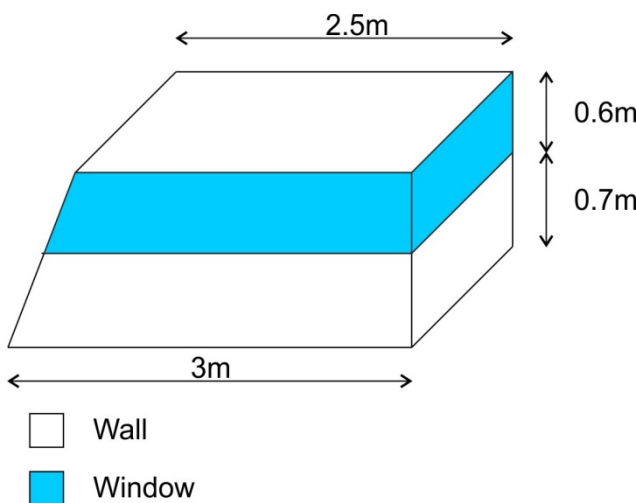


Figure 8: Vehicle geometry

Cabin enclosures are assumed to consist of three layers: 1 mm aluminium, 10 mm mineral wool and 1.2 mm cloth (50% wool, 50% viscose). Fenestration is assumed to be a one-pane window with a transmittance of 0.84 for solar radiation. The vehicle is oriented southwards. Leakages are supposed to lead to one air change per hour (ACH) in the cabin. A ventilation system is running during occupation of the vehicle. This system is assumed to deliver 50 ACH. The supply temperature is controlled to result in a cabin air temperature of 22 °C. Outdoor conditions

are taken from the weather station of Fraunhofer IBP in Holzkirchen, Germany.

Simulation results show a considerable accumulation of water in the insulation (Figure 9). Besides the increased risk of mold growth this leads to increased heat conductivity degrading the performance of the insulation (Figure 10).

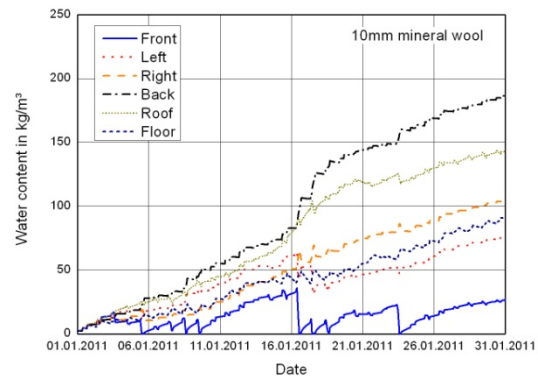


Figure 9: Accumulation of water in the vehicle insulation (10mm)

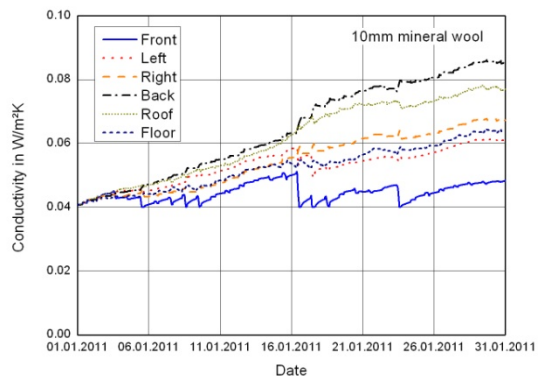


Figure 10: Thermal conductivity of the vehicle insulation (10mm)

To improve the situation the thickness of the insulation can be increased. This leads to higher surface temperature on the cabin side of the insulation resulting in a lower gradient of water vapor pressure thus leading to a lower moisture flow into the insulation. Figure 11 and Figure 12 show moisture content and thermal conductivity when increasing the thickness of mineral wool to 30 mm. The gain of this measure is twofold. A thicker insulation presents a higher resistance to heat. Furthermore, the conductivity of the thicker insulation is lower as less water accumulates.

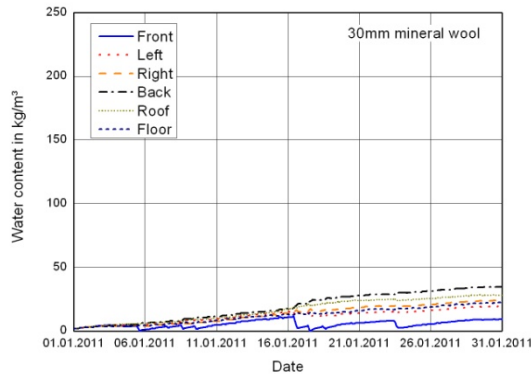


Figure 11 Accumulation of water in the vehicle insulation (30mm)

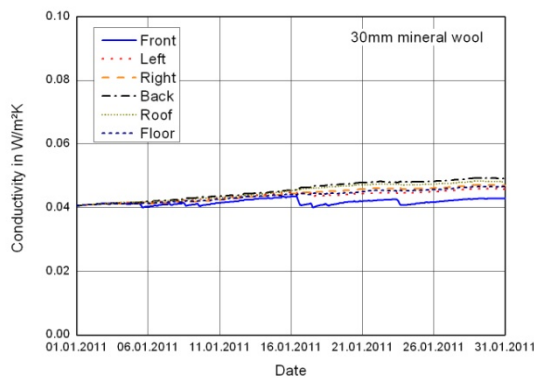


Figure 12: Thermal conductivity of the vehicle insulation (30mm)

5 Ongoing developments

The next step in the development of the Indoor Climate Library is to implement more functionalities than currently available. More detailed radiation models allowing the use of user-defined form factors and templates of predefined form factors for simple generic geometries will be introduced. A database of convective heat and moisture transfer coefficient correlations will be inserted. Interfaces will allow the use of the Air Conditioning Library [10] to model the air supply from HVAC systems. Templates for generic building and vehicle setups will be added. Further application examples will give an overview of the possibilities of the Indoor Climate Library.

6 Conclusion

The Indoor Climate Library allows computing heat and moisture transfer in constructions. A vehicle ap-

plication example shows that the applied usage profile the selected wall layer construction leads to accumulation of moisture in the insulation. The Indoor Climate Library allows quick estimation of remedies to this problem. Increasing the thickness of the insulation reduced water accumulation noticeably.

7 References

- [1] Nouidui, T.: Entwicklung einer objektorientierten Modellbibliothek zur Ermittlung und Optimierung des hygrothermischen und hygienischen Komforts in Räumen, Thesis, Universität Stuttgart, 2008
- [2] LBNL Website, consulted 07. March 2012, <http://simulationresearch.lbl.gov/modelica>
- [3] Wetter, M.: Modelica library for building heating, ventilation and air-conditioning systems, 7th International Modelica Conference, Como, Italy, 20.-22. September 2009
- [4] Wetter, M., Zuo, W., Nouidui, T.: Modeling of heat transfer in rooms in the Modelica "Buildings" Library, Building Simulation, Sydney, Australia, 14.-16. November 2011
- [5] Wetter, M., Zuo, W., Nouidui, T.: Recent developments in the Modelica "Buildings" Library for Building Energy and Control Systems, 8th International Modelica Conference, Dresden, Germany, 20.-22. March 2011
- [6] Sedlbauer, K.: Vorhersage von Schimmelpilzbildung auf und in Bauteilen, Thesis, Universität Stuttgart, 2001
- [7] Fraunhofer IRB: MASEA - Denkmalpflege, consulted on 07. May 2012, http://www.irb.fraunhofer.de/denkmalpflege/angebote_partner/masea/
- [8] Künzel, H.: Verfahren zur ein- und zweidimensionalen Berechnung des gekoppelten Wärme- und Feuchtetransports in Bauteilen mit einfachen Kennwerten, Thesis, Universität Stuttgart, 1994
- [9] Fraunhofer IBP Wufi Website, consulted 07. March 2012, www.wufi.de
- [10] Modelon AB: Flyer of Dymola Air Conditioning Library, 2012, http://www.modelon.com/fileadmin/user_upload/Products/Modelon/ACL/Modelon_ACL_flyer.pdf

Dynamic modelling of a Condenser/Water Heater with the ThermoSysPro Library

Baligh El Hefni **Daniel Bouskela**

EDF R&D

6 quai Watier, 78401 Chatou Cedex, France
baligh.el-hefni@edf.fr daniel.bouskela@edf.fr

Guillaume Gentilini

EDF SEPTEN

12-14 avenue Dutrievoz, 69628 Villeurbanne Cedex, France
guillaume.gentilini@edf.fr

Abstract

A new dynamic model of a water heater has been developed. The component model is meant to be used for power plant modeling and simulation with the ThermoSysPro library developed by EDF and released under open source license.

The model and the test conditions are fully described: modeling hypothesis, governing equations, parameter values and test transients.

To validate the model, three difficult transients were simulated: the islanding (sudden plant disconnection from the grid), flow reversal and zero-flow conditions inside the water heater.

Regarding the islanding scenario, the simulation results are very close to the experimental values measured on site. This transient demonstrates the physical validity of the model at it is fast and challenges the model equations in all operating conditions of the exchanger.

Keywords: *Modelica; thermal-hydraulics ; heat exchanger ; water heater ; dynamic modeling; inverse problems*

1. Introduction

In the framework of the EUROSYSLIB project, a new library called ThermoSysPro has been developed.

The main objective of ThermoSysPro is to provide a generic library for the modeling and simulation of power plants and other kinds of energy systems. The meaning of the word ‘generic’ is here to be taken as

the possibility to use the same library components to model different kinds of energy systems for different types of studies (sizing, control system verification, etc.).

The library is now routinely used for different purposes, see for instance [1 to 6]. An introduction to the library can be found in [5].

New developments are ongoing or planned to extend the scope of the library for uncertainties and state estimation.

The objective of this paper is to show how the library can be extended to include a new component to model a shell-and-tube heat exchanger, by using already existing components of ThermoSysPro.

2. Model of the condenser/water heater

2.1. General presentation of the water heater

The water heater is a **two-phase** shell-and-tube heat exchanger (see Figure 1). The feedwater flows inside the tube bundle, while the steam and condensate flows outside these tubes (inside the cavity). In the water heater, there are three distinct areas: (1) the desuperheating zone, (2) the condensation zone, both located in the upper part of the component, and (3) the subcooled zone, located in the lower part of the component. In some water heaters, the condensate of the water heater located upstream from the current water heater is re-injected into the current water heater. During re-injection, part of the condensate may vaporize due to the pressure drop (this phenomenon is known as flash). The level of the

condensate in the cavity is adjusted with a valve located at outlet of the water heater.

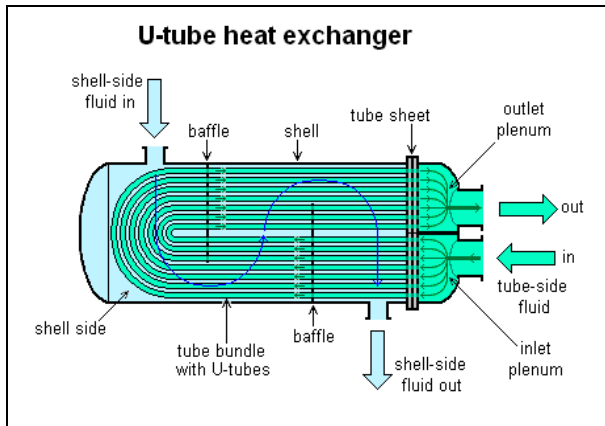


Figure 1: Shell-and-tube heat exchanger

2.2. Description of the water heater model

The **DynamicWaterHeating** model represents the dynamics of the thermo-hydraulic phenomena of the hot fluid inside the cavity and of the cooling fluid which flows through the tube bundle. In particular, the model features the thermal exchanges between the fluid in the cavity and the cooling fluid flowing through the tube bundle.

The water heater is considered as a vertical or horizontal cylindrical cavity (as schematized in Figure 2), containing a U-bent tube bundle with the feedwater inlet and outlet located on the same side. The cavity is subdivided into the following zones:

A) **The desuperheating zone**, where the superheated steam, flowing into the heater, exchanges heat with the liquid flowing through the tube bundle, until it becomes saturated steam and enters the condensation zone. This zone is modelled by ‘Pipe 4’ in Figure 2.

B) **The condensation zone**, where the saturated steam condenses as a consequence of the thermal exchange with the tube bundle, turning into liquid water that enters the subcooled zone. This zone is modelled by ‘Pipe 2’ and ‘Pipe 3’ in Figure 2.

C) **The subcooled zone**, where the liquid inside the cavity continues to exchange heat with the liquid flowing through the tube bundle. This zone is modelled by ‘Pipe 1’ in Figure 2.

Four configurations of the model are possible (see Figure 2):

1. horizontal water heater, with desuperheating zone, condensation zone and subcooled zone,
2. horizontal water heater, with condensation zone only,

3. vertical water heater, with desuperheating zone, condensation zone and subcooled zone,
4. separate vertical water heater with desuperheating zone, condensation zone and subcooled zone.

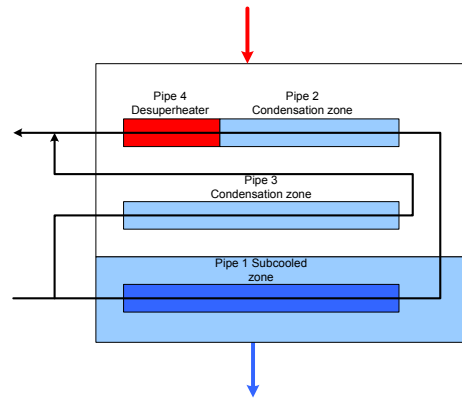


Figure 2a: Horizontal water heater (1)

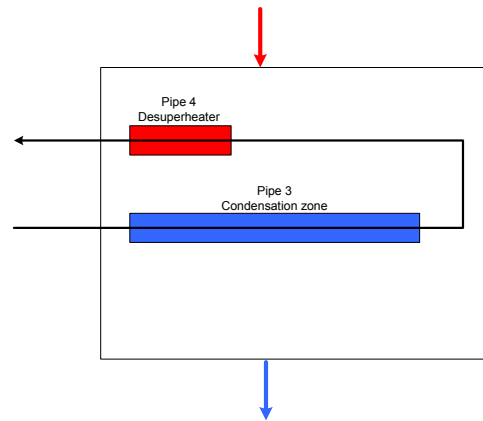


Figure 2b: Horizontal water heater (2)

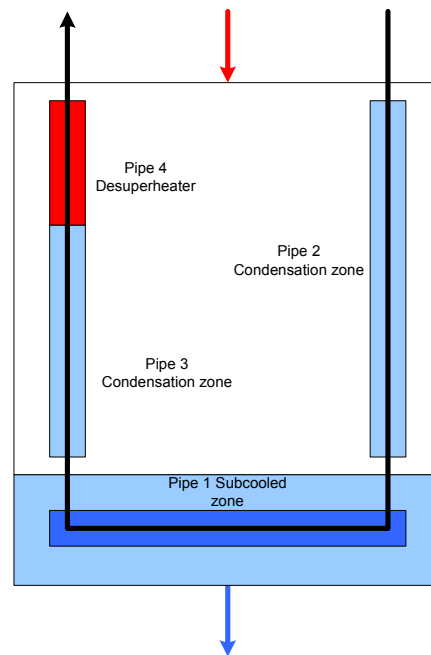


Figure 2c: Vertical water heater (3)

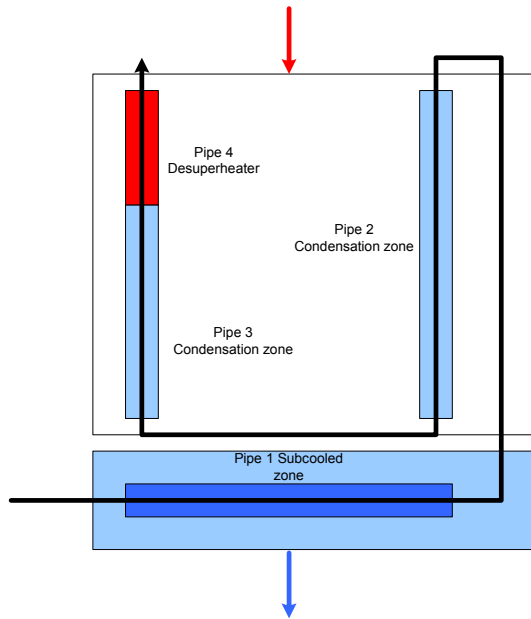


Figure 2d: Vertical separate water heater (4)

2.3. Components of the water heater model

The **DynamicWaterHeating** can simulate all horizontal configurations as shown in Figure 2a and 2b.

The model is divided into sub-models of four different types which are connected together to make the full model (see Figure 3):

- 3 DynamicTwoPhaseFlowPipe models,
- 3 HeatExchangerWall models,
- 1 TwoPhaseCavity model,
- 3 Volume models.

By reassembling the sub-models, any other configuration of the water heater can be modelled.

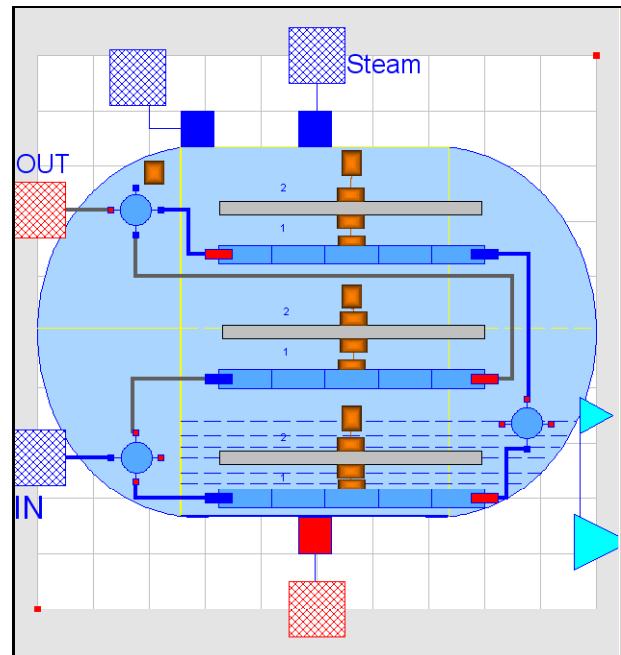


Figure 3: Model of the water heater “DynamicWaterHeating”

The description of each sub-model is given in the following section. Each sub-model in the model can be recognized by looking at its icon (see Figures 4, 5, 6 and 7).

3. Physics of the condenser/water heater

3.1. DynamicTwoPhaseFlowPipe model

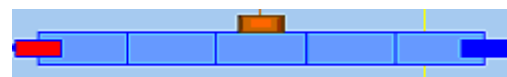


Figure 4: Two-flow pipe model icon

The model of the fluid flow in a cylindrical conduit is based on the dynamic mass, energy, and momentum balance equations, which are originally given as 1-D partial differential equations. The original distributed-parameter model is first discretised by using the finite-volume method. The model is formulated in order to correctly handle possible flow reversal conditions.

Assumptions

- Homogeneous fluid in each mesh cell (same velocity for the liquid and steam phases);
- 1-D modelling (using the finite-volume method);
- The accumulation is considered in each mesh cell;
- The inertia of the fluid is taken into account;
- The phenomenon of longitudinal heat conduction in the metal wall and in the fluid is neglected;
- The thermo-physical properties are calculated on the basis of the average pressure and enthalpy in each mesh cell.

Mass balance equation

The mass balance equation in each cell is given by:

$$A \cdot \frac{d\rho_i}{dt} \cdot \Delta x = \dot{m}_{i-1i} - \dot{m}_{ii+1}$$

Taking the pressure and the specific enthalpy as state variables yields:

$$A \cdot \left[\left(\frac{\partial \rho_i}{\partial P_i} \right)_h \cdot \frac{dP_i}{dt} + \left(\frac{\partial \rho_i}{\partial h_i} \right)_P \cdot \frac{dh_i}{dt} \right] \cdot \Delta x = \dot{m}_{i-1i} - \dot{m}_{ii+1}$$

Energy balance equation

The energy balance equation in each cell is given by:

$$A \cdot \frac{d(\rho_i \cdot u_i)}{dt} \cdot \Delta x = \dot{m}_{i-1i} \cdot h_{i-1i} - \dot{m}_{ii+1} \cdot h_{ii+1} + \Delta W_i$$

with the specific internal energy given by:

$$u_i = h_i - \frac{P_i}{\rho_i}$$

Taking the pressure and the specific enthalpy as state variables yields:

$$A \cdot \left[\left(h_i \cdot \frac{\partial \rho_i}{\partial P_i} - 1 \right) \cdot \frac{dP_i}{dt} + \left(h_i \cdot \frac{\partial \rho_i}{\partial h_i} + \rho_i \right) \cdot \frac{dh_i}{dt} \right] \cdot \Delta x = \dot{m}_{i-1i} \cdot h_{i-1i} - \dot{m}_{ii+1} \cdot h_{ii+1} + \Delta W_i$$

$h_{i,i+1}$ is the specific enthalpy of the mass flow $\dot{m}_{i,i+1}$ crossing the boundary between the cells i and $i+1$. $h_{i,i+1}$ is related to the state variables h_i and h_{i+1} by:

$$h_{i,i+1} = \hat{s}(P_e) \cdot h_i + \hat{s}(-P_e) \cdot h_{i+1}$$

where P_e is the Peclet number and

$$\hat{s}(x) = \frac{1}{1 + e^{-\frac{x}{2}}} \quad (\text{see e.g. [8]}).$$

When neglecting diffusion, the Peclet number is infinite, and

$$h_{i,i+1} = s(\dot{m}_{i,i+1}) \cdot h_i + s(-\dot{m}_{i,i+1}) \cdot h_{i+1}$$

where s is the step function:

$$s(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x < 0 \end{cases}$$

This simplification is known as the upwind scheme.

Momentum balance equation

The momentum balance equation in each cell is given by:

$$1/A \cdot \frac{d\dot{m}_{i,i+1}}{dt} \cdot \Delta x =$$

$$P_i - P_{i+1} - (\Delta P)_{i,i+1}^a - (\Delta P)_{i,i+1}^f - (\Delta P)_{i,i+1}^g$$

with respectively the acceleration, friction and gravity pressure losses given by:

$$(\Delta P)_{i,i+1}^a = \frac{1}{A^2} \cdot \dot{m}_{i,i+1} \cdot |\dot{m}_{i,i+1}| \cdot \left(\frac{1}{\rho_{i+1}} - \frac{1}{\rho_i} \right)$$

$$(\Delta P)_{i,i+1}^f = \zeta \cdot \frac{\Lambda_i \cdot \Delta x_h}{2 \cdot D \cdot A^2 \cdot \rho_i} \cdot \dot{m}_{i,i+1} \cdot |\dot{m}_{i,i+1}|$$

$$(\Delta P)_{i,i+1}^g = \rho_{i,i+1} \cdot g \cdot (z_{i+1} - z_i)$$

By default, the flow is considered turbulent (Reynolds number $Re > 2300$).

The **Colebrook** correlation is used to compute Λ_i .

Convective heat transfer within the U-tubes

The heat exchanged between the fluid and the wall is:

$$\Delta W(i) = h_c(i) \cdot \Delta S_2 \cdot (T_{w2}(i) - T(i))$$

Convection heat transfer coefficient

The convection heat transfer coefficient h_c between the fluid and the wall is computed using the **Dittus-Boelter** correlation.

3.2. HeatExchangerWall model

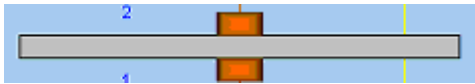


Figure 5: Wall model icon

The wall model describes the conductive heat flow through the wall of the tube bundle. The flow is positive when entering the tubes (going from side 2 to side 1 of the wall).

$$\Delta W_1(i) = \frac{2 \cdot \pi \cdot \lambda \cdot \Delta x(i) \cdot ntubes \cdot (T_w(i) - T_{w1}(i))}{\ln((e + D)/D)}$$

$$\Delta W_2(i) = \frac{2 \cdot \pi \cdot \lambda \cdot \Delta x(i) \cdot ntubes \cdot (T_{w2}(i) - T_w(i))}{\ln((2 \cdot e + D)/(e + D))}$$

$$\Delta M_w \cdot c_{pw} \cdot \frac{dT_w}{dt} = \Delta W_2(i) - \Delta W_1(i)$$

3.3. TwoPhaseCavity model

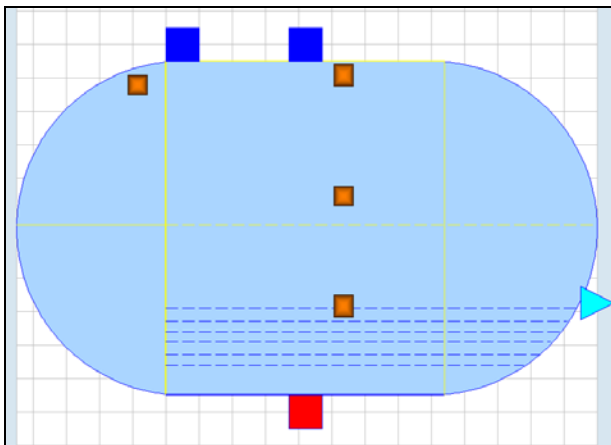


Figure 6: Two-phase cavity model icon

The cavity is modelled as a non-adiabatic two-phase volume, with vertical or horizontal cylindrical geometry. The physical model is based on a non-equilibrium, two-phase formulation of the fluid balance equations with a control volume approach. The two phases are supposed to be isobaric and will be referred to as liquid zone and steam zone, respectively.

The model features the condensation flow of the steam phase into the liquid phase, and reciprocally, the vaporization flow of the liquid phase into the steam phase.

The reasons for not assuming thermal equilibrium between the two phases are:

- The vapour may enter the cavity superheated (the vapour temperature is then higher than the saturation temperature).
- The liquid may be subcooled by the incoming drain and the wetted tube bundle (the liquid temperature is then lower than the saturation temperature).

Assumptions

- Accumulation of mass and energy is considered. Heat exchange between the liquid and steam phases is considered.
- Heat exchange between the liquid or steam phases and the wall is considered.
- Heat exchange between the water heater and the external medium (ambient) is considered.
- Pressure losses are not taken into account in the cavity.
- The liquid and steam phases are not necessarily in thermal equilibrium.
- The liquid and steam phases are assumed to be permanently in pressure equilibrium.

State variables

The state variables of the system are:

- the mean pressure in the cavity,
- the specific enthalpy of the liquid phase,
- the specific enthalpy of the steam phase,
- the temperature of the wall,
- the volume of the liquid phase.

The volume of the steam phase is bound to the volume of the liquid phase by the following equation:

$$V_l + V_v = V$$

Mass balance equation in each phase

$$\frac{d(\rho_l \cdot V_l)}{dt} = -\dot{m}_l^o + (1 - x_{mv}) \cdot \dot{m}_{drain}^e + \dot{m}_{cond} - \dot{m}_{evap}$$

$$\frac{d(\rho_v \cdot V_v)}{dt} = \dot{m}_v^e + x_{mv} \cdot \dot{m}_{drain}^e + \dot{m}_{evap} - \dot{m}_{cond}$$

where \dot{m}_v^e is the mass flow of incoming vapor, \dot{m}_{drain}^e is the mass flow of the incoming condensate of the water heater located upstream, \dot{m}_l^o is the mass flow of outgoing condensate, \dot{m}_{cond} is the condensation flow inside the cavity, and \dot{m}_{evap} is the evaporation flow inside the cavity.

Condensation and evaporation mass flow rate inside the cavity

$$\dot{m}_{cond} = \begin{cases} x_v < X_{vo} \Rightarrow C_{cond} \cdot \rho_v \cdot V_v \cdot (X_{vo} - x_v) \\ x_v \geq X_{vo} \Rightarrow 0 \end{cases}$$

$$\dot{m}_{evap} = \begin{cases} x_l > X_{lo} \Rightarrow C_{evap} \cdot \rho_l \cdot V_l \cdot (x_l - X_{lo}) \\ x_l \geq X_{lo} \Rightarrow 0 \end{cases}$$

C_{cond} and C_{evap} being coefficients with inverse time dimensionality [$\frac{1}{t}$], X_{vo} and X_{lo} denoting constants.

Energy balance equation in each phase

The general form of the energy balance equation is given by:

$$\frac{d(\rho \cdot V \cdot u)}{dt} = \sum_e \dot{m}_e \cdot h_e + \sum_o \dot{m}_o \cdot h_o + \sum W$$

Taking the pressure and the specific enthalpy as state variables yields:

$$V_l \cdot \left[\left(\frac{P}{\rho_l} \cdot \left(\frac{\partial \rho_l}{\partial P} \right)_h - 1 \right) \cdot \frac{dP}{dt} + \left(\frac{P}{\rho_l} \cdot \left(\frac{\partial \rho_l}{\partial h_l} \right)_p + \rho_l \right) \cdot \frac{dh_l}{dt} \right] =$$

$$-\dot{m}_l^o \cdot \left(h_l^o - \left(h_l - \frac{P}{\rho_l} \right) \right) + \dot{m}_{cond} \cdot \left(h_l^{sat} - \left(h_l - \frac{P}{\rho_l} \right) \right)$$

$$-\dot{m}_{evap} \cdot \left(h_v^{sat} - \left(h_l - \frac{P}{\rho_l} \right) \right) + (1 - x_{mv}) \cdot \dot{m}_{drain}^e \cdot \left(h_{drain}^e - \left(h_l - \frac{P}{\rho_l} \right) \right)$$

$$+ W_{vl} - W_{lw} - W_{lt}$$

$$V_v \cdot \left[\left(\frac{P}{\rho_v} \cdot \left(\frac{\partial \rho_v}{\partial P} \right)_h - 1 \right) \cdot \frac{dP}{dt} + \left(\frac{P}{\rho_v} \cdot \left(\frac{\partial \rho_v}{\partial h_v} \right)_p + \rho_v \right) \cdot \frac{dh_v}{dt} \right] =$$

$$\dot{m}_v^e \cdot \left(h_v^e - \left(h_v - \frac{P}{\rho_v} \right) \right) - \dot{m}_{cond} \cdot \left(h_l^{sat} - \left(h_v - \frac{P}{\rho_l} \right) \right)$$

$$+ \dot{m}_{evap} \cdot \left(h_v^{sat} - \left(h_v - \frac{P}{\rho_v} \right) \right) + x_{mv} \cdot \dot{m}_{drain}^e \cdot \left(h_{drain}^e - \left(h_v - \frac{P}{\rho_l} \right) \right)$$

$$- W_{vl} - W_{vw} - W_{2t} - W_{3t}$$

Energy accumulation at the wall

$$M_w \cdot c_{pw} \cdot \frac{dT_w}{dt} = W_{lw} + W_{vw} - W_{wa}$$

Heat exchange between the liquid and steam phases

$$W_{vl} = K_{vl} \cdot A_{vl} \cdot (T_v - T_l)$$

Heat exchange between the liquid or steam phases and the wall

$$W_{lw} = K_{lw} \cdot A_{lw} \cdot (T_l - T_w)$$

$$W_{vw} = K_{vw} \cdot A_{vw} \cdot (T_v - T_w)$$

Heat exchange between the water heater and the external medium

$$W_{wa} = K_{va} \cdot A_{va} \cdot (T_v - T_a)$$

In this equation, the vapor temperature is considered instead of the wall temperature to account for both the thermal resistance of the metallic wall and the thermal insulator of the cavity, in addition to the usual convective resistance. Consequently, K_{va} is the global heat exchange coefficient between the vapor and the ambient. The liquid is neglected in this equation because the volume of liquid is small w.r.t. the volume of vapor.

Heat exchange between the liquid and the tube bundle 'Pipe 1'

$$\Delta W_1(i) = h_{conv1}(i) \cdot \Delta S_{ext1} \cdot (T_l - T_{w1}(i))$$

$$W_{1t} = \sum \Delta W_1(i)$$

Heat exchange between the steam and the tube bundle 'Pipe 2'

$$\Delta W_2(i) = h_{cond2}(i) \cdot \Delta S_{ext2} \cdot (T_v - T_{w2}(i))$$

$$W_{2t} = \sum \Delta W_2(i)$$

Heat exchange between the steam and the tube bundle 'Pipe 3'

$$\Delta W_3(i) = h_{cond3}(i) \cdot \Delta S_{ext3} \cdot (T_v - T_{w3}(i))$$

$$W_{3t} = \sum \Delta W_3(i)$$

Heat exchange between the steam and the tube bundle 'Pipe 4'

$$W_{4t} = \dot{m}_v^e \cdot (h_v^e - h_v^{sat})$$

Heat transfer convection coefficients

The heat transfer convection coefficient h_{conv} between the water and the outside wall of the tube bundle is computed using the **Kern** correlation [7].

The **Nusselt** correlation is used to calculate the heat transfer coefficients h_{cond} between the steam and the outside wall of the tube bundle, in the condensation zone.

3.4. Mixture homogeneous Volume model

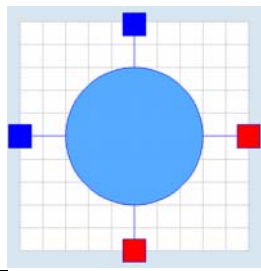


Figure 7: Mixing volume model icon

This sub-model describes the mixing of one-phase flow fluid.

Mass balance equation

$$\frac{d(\rho \cdot V)}{dt} = \sum_e \dot{m}_e + \sum_o \dot{m}_o$$

Energy balance equation

$$\frac{d(\rho \cdot V \cdot u)}{dt} = \sum_e \dot{m}_e \cdot h_e + \sum_o \dot{m}_o \cdot h_o + \sum W$$

4. Validation of the condenser/water heater

4.1. Modelica model of the condenser/water heater

To simulate the complex dynamic physical behaviour in normal and accidental conditions of the condenser/water heater model, a test model called “TestDynamicWaterHeating” has been developed by assembling the necessary components from the **ThermoSysPro** library (cf. Figure 8). The test model includes the level control system.

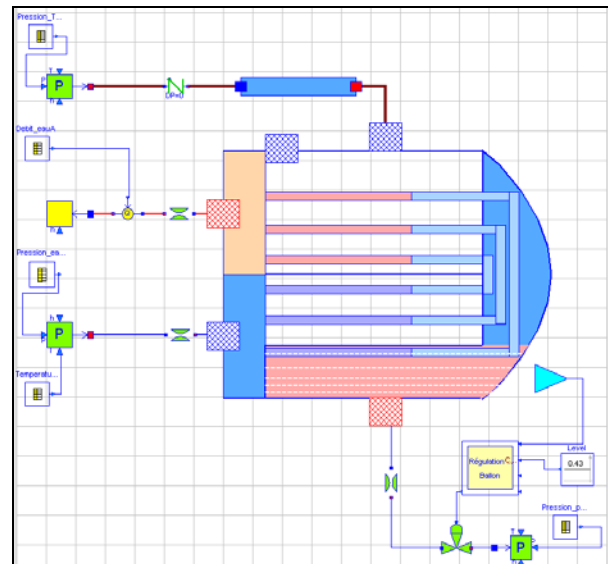


Figure 8: Model of the water heater “TestDynamicWaterHeating”

4.2. Data implemented in the model

All geometrical data were provided to the model (tubes and exchangers lengths, diameters, volumes, corrective terms for the heat exchange coefficients, corrective terms for the pressure losses, etc.). The plant characteristics are given in Figure 11 (cf. Appendix).

4.3. Calibration of the model

The calibration phase consists in setting (blocking) the maximum number of thermodynamic variables to known measurement values (enthalpy, pressure) taken from on-site sensors for 100% load. This method ensures that all needed performance parameters, size characteristics and output data can be computed.

The main computed performance parameters are:

- the correction coefficient of the heat transfer coefficient inside the condensation zone,
- the correction coefficient of the pressure loss coefficients inside the tube bundle (pipes),
- the pressure loss coefficients of the pipeline between the steam turbine and the water heater,
- the maximum Cv values of the extraction valve and the valves positions.

4.4. Simulation scenario: islanding

In order to challenge the dynamic simulation capabilities of the model, a high amplitude transient, called islanding, that occurs when the plant is suddenly disconnected from the normal energy dis-

charge network, is simulated. This transient is used to check and validate the physics taken into account in the model and the numerical robustness of the model as it runs the water heater model into very different operating regimes. This allows to test the validity and applicability range of the model equations, and the numerical robustness of the Modelica implementation when using Dymola.

4.5. Boundary conditions of the model

The boundary conditions of the model (scenario profiles) are presented in Figure 9.

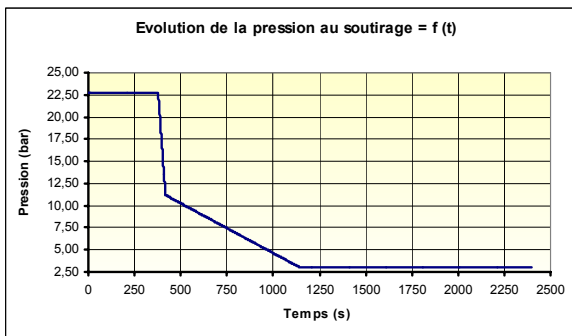


Figure 9a: Outlet pressure of the steam turbine

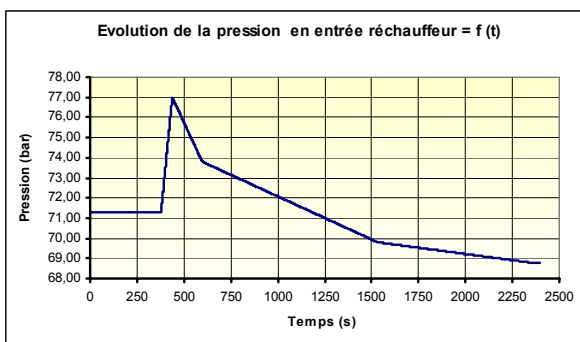


Figure 9b: Inlet pressure of the feed water

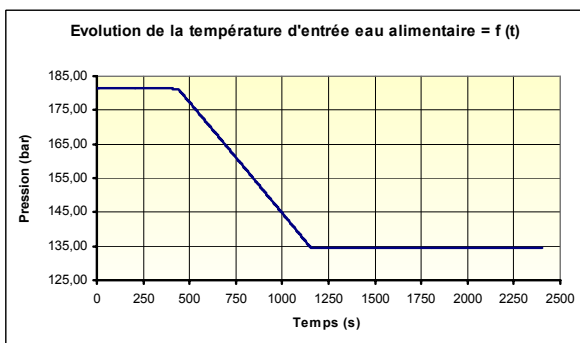


Figure 9c: Inlet temperature of the feed water

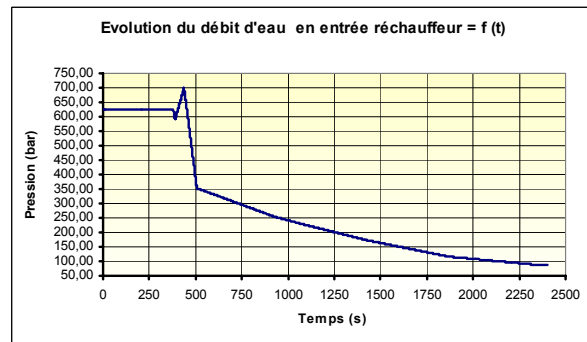


Figure 9d: Inlet flow of the feed water

4.6. Results of dynamic simulations

In order to cover the whole transient, the simulation time has been set at 2500 seconds.

Simulation runs were done using Dymola 6.1. The simulation of the scenarios were mostly successful, with only one iteration variable to be fed manually.

The following phenomena are simulated:

- flow reversal,
- local boiling or condensation,
- swell and shrink effect in cavity,
- cavity levels and cavity pressure control.

The model is able to compute precisely:

- the mass flow rate of the steam (at the inlet),
- the mass flow rate of the condensate (drain),
- the distribution of water and steam mass flow rate inside the tubes,
- the thermal power of the water heater and tubes,
- the pressure temperature and specific enthalpy distribution across the network,
- the cavity levels and and cavity pressure.

The results of the simulation runs are given in Figure 10. Figures 10a and 10b show that the results obtained with Dymola are very close to the measured values on site. The outflow drain (condensate) in Figure 10d depends on the way the level is controlled inside the heater.

So, the physical validity of the component model is demonstrated, because we believe that this type of fast transient is likely to extensively validate the physics inside the model as it challenges the water heater in very different operating regimes of the rated operation.

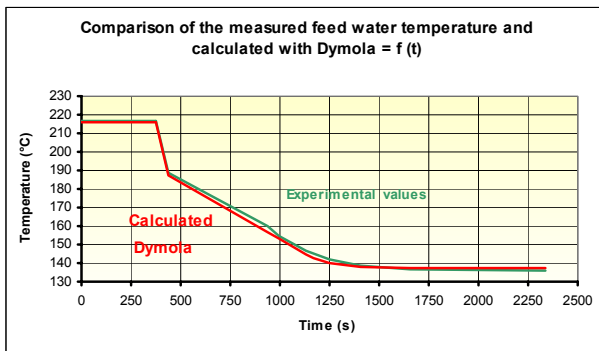


Figure 10a: Evolution of the feed water outlet temperature

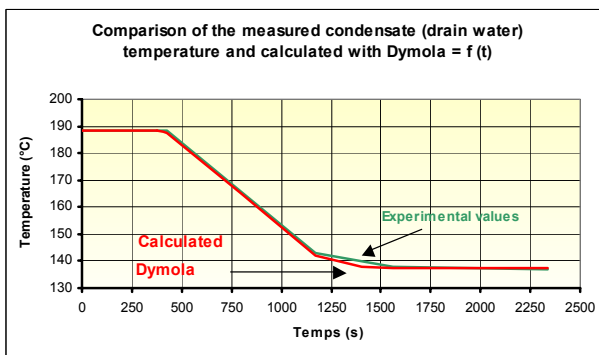


Figure 10b: Evolution of the condensate (water drain) outlet temperature

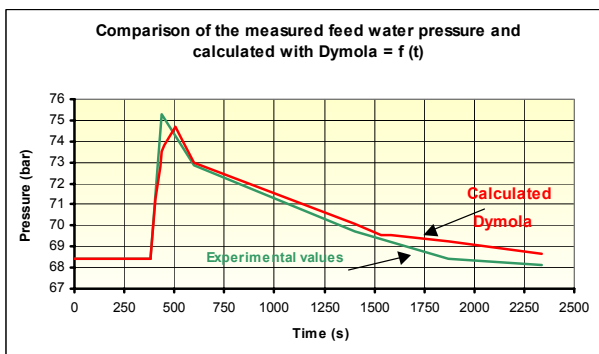


Figure 10c: Evolution of the feed water outlet pressure

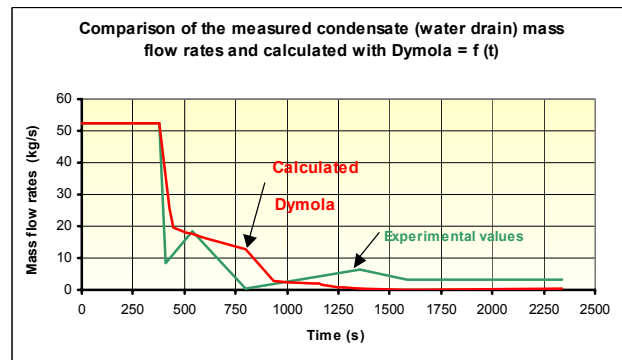


Figure 10d: Evolution of the condensate (water drain) outlet mass flow rate

4.7. Validation of the water heater model under flow reversal and zero-flow conditions

The ThermoSysPro library handles flow reversals.

The boundary conditions for the flow reversal scenario are:

- outlet pressure of the steam turbine = $22.733e5$ Pa,
- outlet enthalpy of the steam turbine = $2650.6e3$ J/kg,
- inlet pressure of feed water = $71.29e5$ Pa,
- inlet temperature of the feed water = 454.46 °C,
- inlet mass flow rate of the feed water ($t = 0$) = 624.97 kg/s,
- inlet mass flow rate of the feed water ($t > 2000s$) = -200 kg/s,
- outlet enthalpy of the feed water inlet ($Q < 0$) = $940.e3$ J/kg.

Figures 12 and 13 in the Appendix show the results for the scenario of flow reversal in the water heater and the results for the zero-flow scenario.

The possibility of flow reversal and zero-flow in the tube bundle of the component has been experimentally verified. But there are no data available for comparison with the simulation results.

5. Conclusion

A new open source Modelica library called 'ThermoSysPro' has been developed within the framework of the ITEA 2 EUROSYSLIB project. This library has been mainly designed for the static and dynamic modeling of power plants, but can also be used for other energy systems such as industrial processes, buildings, etc. It is intended to be easily understood and extendable by the models developer.

A new dynamic model of a water heater has been developed using existing elements of ThermoSysPro.

To validate the model, three difficult transients were simulated: the islanding (sudden plant disconnection from the grid), flow reversal and zero-flow inside the water heater.

Regarding the islanding scenario, the simulation results obtained with Dymola are very close to the experimental values measured on site. This transient demonstrates the physical validity of the model at it is fast and challenges the model equations in all operating conditions of the exchanger.

The possibility of flow reversal and zero-flow occurring inside the tube bundle of the module has been experimentally verified and simulated, but no experimental data is available for comparison with the simulation results.

Nomenclature

Symbols

\dot{m}	Mass flow
ρ	Fluid density
h	Fluid specific enthalpy
u	Fluid specific internal energy
P	Fluid pressure
T	Fluid temperature
c_p	Fluid specific heat capacity
V	Volume
t	Time
W	Power
x_v	Vapor mass fraction in vapor phase
x_l	Vapor mass fraction in liquid phase
x_{mv}	Vapor mass fraction in input drain
Λ	Friction coefficient
ζ	Friction corrective coefficient
Δx	Tube segment length
ΔS	Heat surface exchange of tube segment
D	Tube diameter
A	Tube cross section or heat exchange surface
e	Wall thickness
λ	Conduction coefficient
K	Heat exchange coefficient
M	Mass
h_c	Convective coefficient

h_{conv1}	Convective coefficient of heat transfer between the condensate and the tube bundle in Pipe 1.
h_{cond2}	Convective coefficient of heat transfer by condensation between the vapor and the tube bundle in Pipe 2.
h_{cond3}	Convective coefficient of heat transfer by condensation between the vapor and the tube bundle in Pipe 3.
$ntubes$	Number of tubes in the bundle

Indices

X_i or $X(i)$	Quantity in volume i
X_{i+1}	Flow between volume i and $i+1$
X_e or X^e	Quantity at inlet
X_o or X^o	Quantity at outlet
X_l	Quantity relative to liquid
X_v	Quantity relative to vapor
X_w	Quantity relative to wall
X_{ext}	Quantity relative to external side of wall
X_a	Quantity relative to ambient
X^{sat}	Quantity relative to saturated phase
X_{cond}	Quantity relative to condensation
X_{evap}	Quantity relative to evaporation
X_{drain}	Quantity relative to drain (condensate)
X_1	Quantity relative to Pipe 1
X_2	Quantity relative to Pipe 2
X_3	Quantity relative to Pipe 3
X_4	Quantity relative to Pipe 4

References

- [1] Bouskela D., Chip V., El Hefni B., Favennec J.M., Midou M. and Ninet J. 'New method to assess tube support plate clogging phenomena in steam generators of nuclear power plants', Mathematical and Computer Modelling of Dynamical Systems, 16: 3, 257-267, 2010.
- [2] El Hefni B., Bouskela D., 'Modelling of a water/steam cycle of the combined cycle power plant "Rio Bravo 2" with Modelica', Modelica 2006 conference proceedings.

- [3] David F., Souyri A., Marchais G., 'Modelling Steam Generators for Sodium Fast Reactors with Modelica', Modelica 2009 conference proceedings
- [4] El Hefni B., Péchiné B., 'Model driven optimization of biomass CHP plant design', Mathmod conference 2009, Vienna, Austria.
- [5] El Hefni B., Bouskela D., 'Dynamic modelling of a combined cycle power plant with ThermoSysPro', Modelica 2011 conference proceedings
- [6] Souyri A., Bouskela D., 'Pressurized Water Reactor Modelling with Modelica', Modelica 2006 conference proceedings.
- [7] Collier J.G., and Thome J.R., 'Convective Boiling and Condensation', McGraw-Hill Book Company (UK) limited, 1972 Clarendon Press, Oxford, 1996.
- [8] Patankar S.V., 'Numerical Heat Transfer and Fluid Flow', Hemisphere Publishing Corporation, Taylor & Francis, 1980.

Appendix

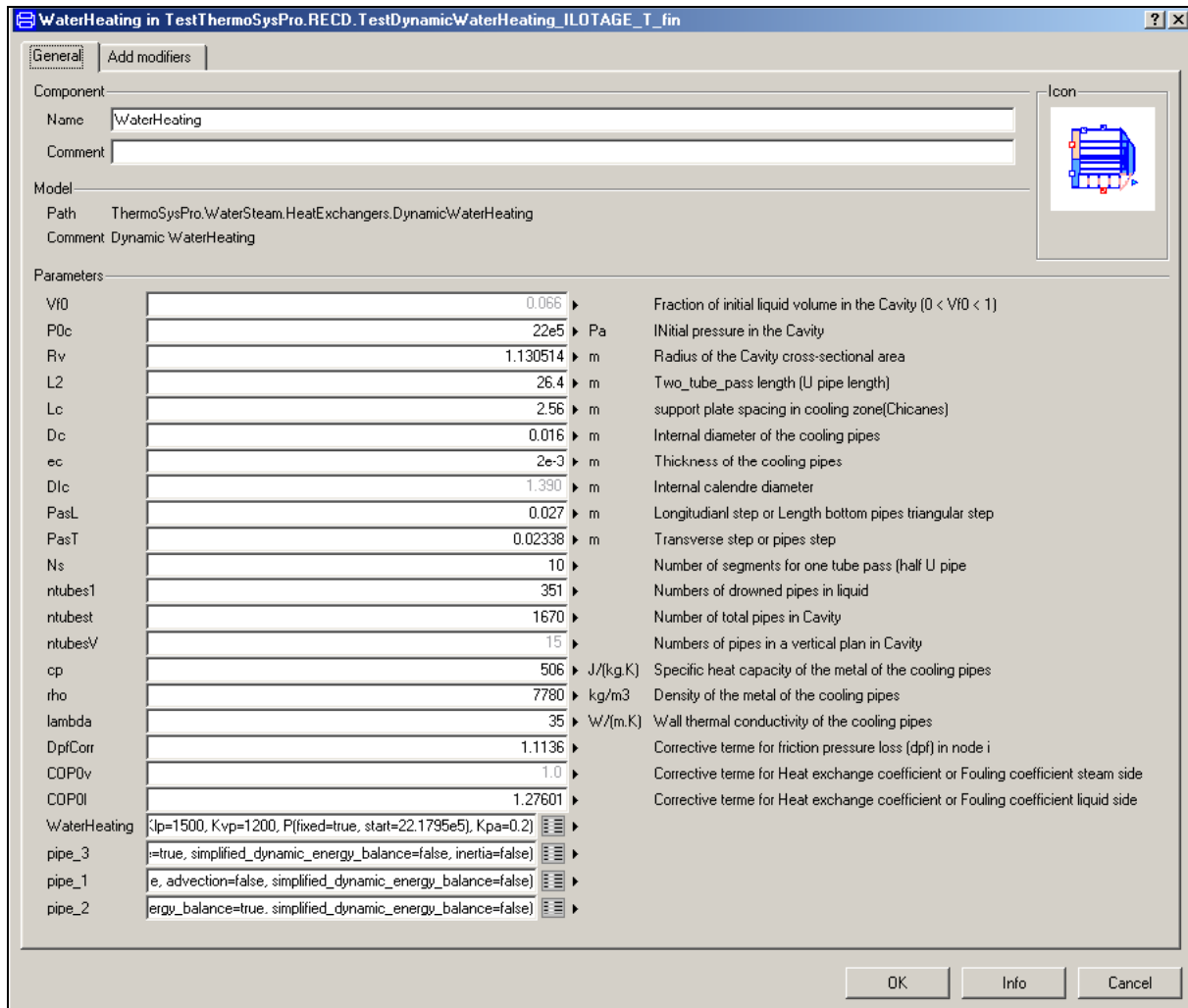


Figure 11: Data of the model

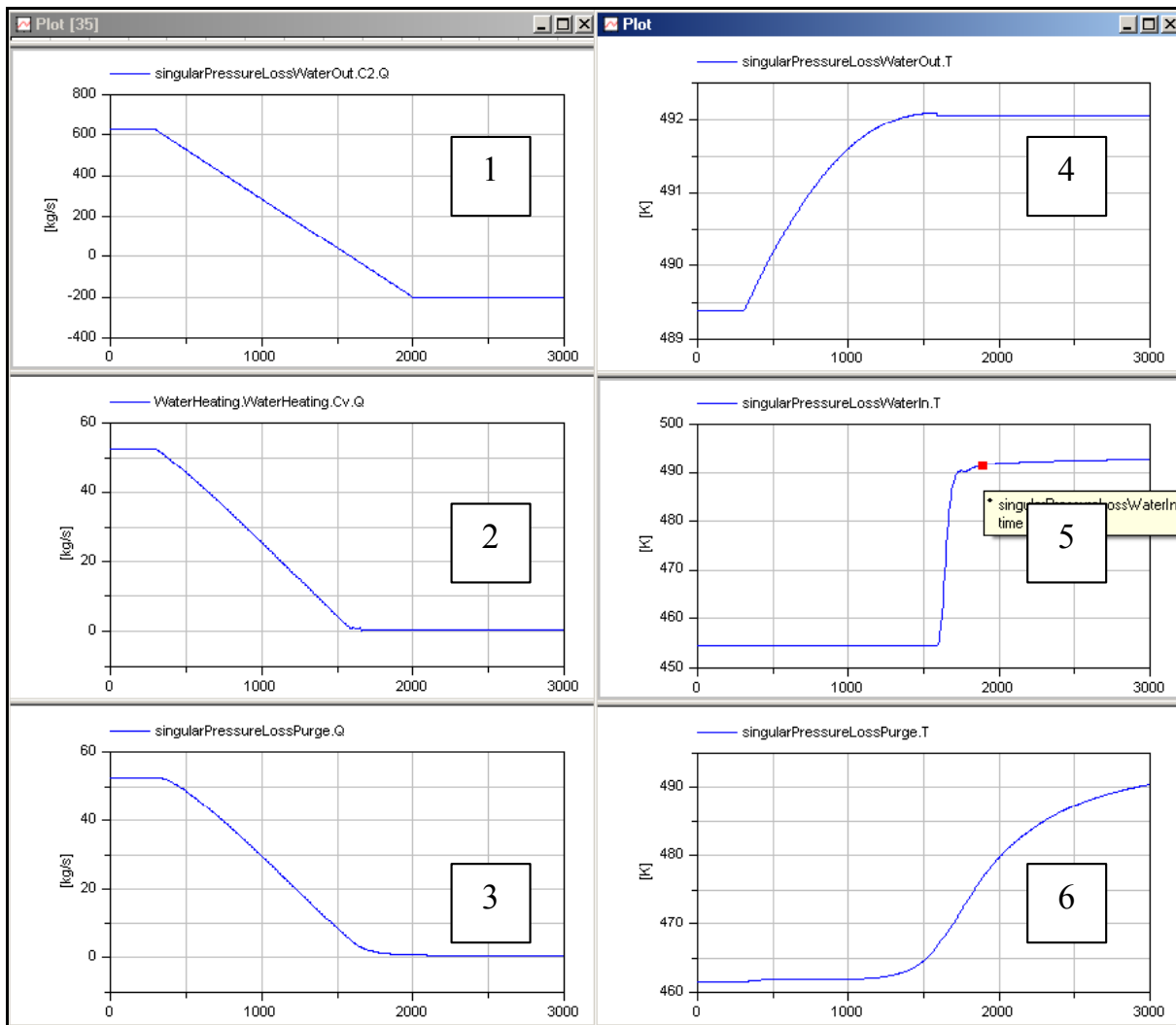


Figure 12: Results for the flow reversal scenario

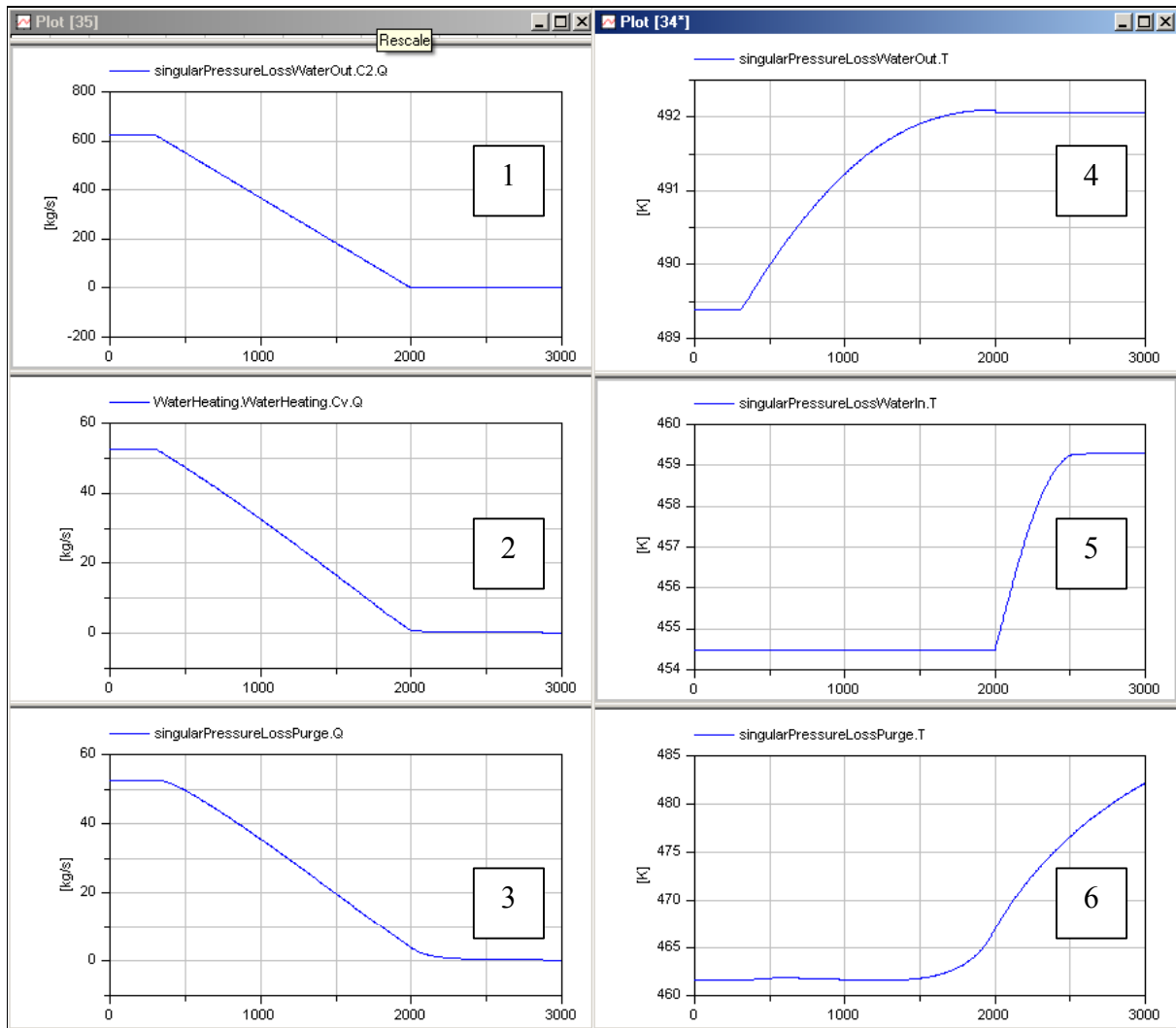


Figure 13: Results for the zero-flow scenario

With:

- 1 - Evolution of the inlet mass flow rate of the feed water,
- 2 - Inlet mass flow rate of the steam (corresponding to the steam turbine outlet),
- 3 - Outlet mass flow rate of the water (output drain),
- 4 - Outlet temperature of the feed water (pipes),
- 5 - Inlet temperature of the feed water (pipes),
- 6 - Outlet temperature of the water (output drain).

FMI implementation in LMS Virtual.Lab Motion and application to a vehicle dynamics case

Hunor Erdélyi, William Prescott, Stijn Donders, Jan Anthonis
LMS international
Interleuvenlaan 68 - 3001 Leuven - Belgium
hunor.erdelyi@lmsintl.com

Abstract

The aim of this paper is to present the implementation of the Modelisar Functional Mock-up Interface (FMI) in LMS Virtual.Lab Motion. This functionality enables co-simulation between multi-disciplinary subsystem models for a range of industrial applications. The validity of the methodology and industrial applicability of the implementation is demonstrated on an application case taken from automotive industry, with an Opposite Wheel Travel scenario using a half vehicle model in LMS Virtual.Lab Motion and an Air-spring FMU based on Modelica code.

Keywords: Functional Mock-up Interface (FMI); Modelica; Co-simulation; LMS Virtual.Lab Motion

1 Introduction

In complex systems such as in automotive and aerospace many different types of subsystems (e.g. mechanical, hydraulic or electric subsystems) interact with each other [1]. The simulation of such complex multidisciplinary systems is a new challenge in modern computer aided engineering.

A widely used technique to link together different multidisciplinary subsystems in a common simulation framework is what scientific literature refers to as *Co-Simulation*. In co-simulation, the overall system is split into different subsystems, which are treated by different optimized simulation tools, coupled by input and output variables, thus creating a coupling loop [2, 3].

The “Functional Mock-up Interface” (FMI) [4], developed within the framework of the ITEA2 Modelisar project [5], provides a standardized way for linking together different subsystems modeled in different simulation software. An instance of a model compiled for being linked with a 3rd party simulation environment is called a “Functional Mock-up Unit” (FMU).

Typically an FMU consists of the following main elements compressed into a single archive:

- a) C-header files to interact with the equations of a model or to perform co-simulations with other simulators (model interface) and
- b) XML schema files to inquire information about model and interface variables (model description file)
- c) executable files

Two distinct standards have been defined within the framework of FMI: *FMI for Model Exchange* and *FMI for Co-Simulation*. The FMI for **Model Exchange** was developed to allow a modeling tool to generate C code or binary files from a model that can be integrated into another simulation environment [4]. The FMI for **Co-Simulation** defines an interface standard for the communication between a master and the individual simulation tools called slaves in a co-simulation environment. The data exchange is restricted to discrete communication points in time and the subsystems are solved independently between these communication points [4, 6].

FMI compatibility was implemented in LMS Virtual.Lab Motion [7], a multi-purpose simulation software, specially designed to simulate realistic motion and loads of mechanical system. LMS Virtual.Lab Motion can be used as a simulation platform into which one or several FMUs can be linked in order to perform simulations for analyzing complex multidisciplinary systems.

2 FMI Interface in LMS Virtual.Lab Motion

A schematic representation of linking an FMU into a simulation with LMS Virtual.Lab Motion is presented in Figure 1. To be able to establish the link between LMS Virtual.Lab Motion and an FMU, *inputs* and *outputs* have to be defined, which will represent the coupling data for the co-simulation.

The coupling data is exchanged at the level of **Control Nodes**. A **Control Input** represents the signal which is transmitted from the mechanical model in LMS Virtual.Lab Motion to the FMU. Typically, Control Inputs are displacement, velocity or acceleration data. A **Control Output** is a signal received from an FMU that is applied to the mechanical model in LMS Virtual.Lab Motion (e.g. force or torque). **Control Nodes** are the nodes or connection points to which the above mentioned Control Inputs and Outputs are applied.

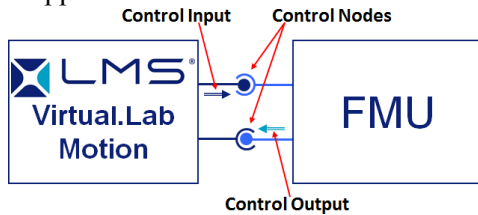


Figure 1: Schematic representation of the FMI interface in LMS Virtual.Lab Motion

For the two distinct standards, FMI for Model Exchange and FMI for Co-Simulation, the different approaches are described as follows.

In case of linking to an FMU for Model Exchange the state equations of both the FMU and LMS Virtual.Lab Motion are solved by the Motion solver.

The LMS Virtual.Lab Motion Solver uses a set of Differential-Algebraic equations (DAE) of motion in Newton-Euler format [7].

$$M\dot{v} + \Phi_q^T \lambda = Q_a(q, v) \quad (1)$$

$$\Phi(q) = 0 \quad (2)$$

Here, q is the vector of generalized position coordinates, v denotes the vector of generalized coordinate velocities, M is the mass matrix, Q_a is the vector of applied forces, $\Phi(q)$ denotes the vector joint constraint equations and λ stands for the vector of Lagrange multipliers. A maximal set of coordinates are considered first and then the extra degrees of freedom are removed by applying a set of joint constraint equations.

When linking an FMU for Model Exchange to LMS Virtual.Lab Motion a set of control forces is applied on the mechanism bodies representing the contribution of the FMU. In turn sensors feed position, velocity and acceleration data back to the FMU. Usually, the FMU forces are the product of state equations. This means that the Motion solver must integrate a set of differential equations from the FMU.

Representing the FMU state equations by g and the state variable by χ , the coupled equations of motion become:

$$M\dot{v} + \Phi_q^T \lambda = Q_a(q, v, \chi) \quad (3)$$

$$\Phi(q) = 0 \quad (4)$$

$$g(q, v, \chi, \dot{\chi}) = 0 \quad (5)$$

In case of linking to an FMU for Co-Simulation, each simulation package runs its own solver, which is in turn synchronized with the other solver. Each solver is running and communicating with the other solver at discrete intervals in time. The same equations (3-5) are solved in the co-simulation mode as in the case of model exchange, but separately. In this situation the LMS Virtual.Lab Motion solver is the master. The Motion solver solves its own set of state equations from the current time (t^i) to the time at the next communication interval (t^{i+1}). Equation (5) now becomes equation (6) where the FMU variable inputs (q, v) are still at the last sample time.

$$g(q^i, v^i, \chi, \dot{\chi}) = 0 \quad (6)$$

Once the LMS Virtual.Lab Motion solver has finished integrating to the next communication interval the FMU solver is called and told to integrate to the current time. The FMU solver now uses the LMS Virtual.Lab Motion inputs at the last communication interval to move forward to the next communication interval.

$$M\dot{v} + \Phi_q^T \lambda = Q_a(q, v, \chi^i) \quad (7)$$

$$\Phi(q) = 0 \quad (8)$$

For both cases described above, a fixed communication interval has been used.

In the following paragraphs, the implementation of the FMI standard into LMS Virtual.Lab Motion will be demonstrated with a simple air-spring FMU.

3 Application case description and results

For demonstrating the implementation of the FMI interface and industrial applicability, an application case is presented from automotive industry, with an Opposite Wheel Travel scenario using a half vehicle model in LMS Virtual.Lab Motion and an Air-spring FMU based on Modelica code.

3.1 Development of a Modelica FMU of an air-spring

An air-spring can be approximated as a volume of air, enclosed either in a cylinder fitted with a piston or in a flexible bellows, as shown in Figure 2. The air is compressed to a predetermined pressure under the static load of the vehicle. Subsequent motion of the piston either increases or decreases the pressure and consequently increases or decreases the force acting on the piston.

For simplicity, the air-spring is modeled with an isothermal process, considering a closed system and

ideal gas. The chamber of the gas is considered as rigid, thus neglecting the elasticity of the bellow.

The diameter of the piston is variable as highlighted in Figure 2.

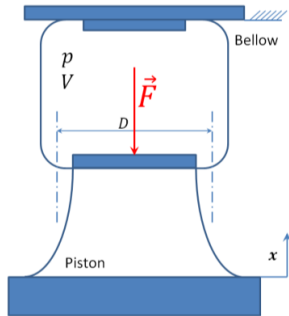


Figure 2: Schematic representation of an air-spring (p is the pressure and V is the volume of the gas, D represents the piston diameter and F the piston force, x is the piston displacement)

For an ideal gas at constant temperature, the Boyle-Mariotte law is valid (9):

$$pV = nRT = \text{constant} \quad (9)$$

Where, p denotes the pressure of the system, V denotes the volume of the gas, n is the number of moles of gas present, R is the ideal gas constant and T denotes the temperature of the system.

Considering the air-spring modeled as an isothermal process, the pressure p of the system will be variable as a function of the volume V . Furthermore, the volume V depends on the displacement and diameter of the piston of the air-spring.

The diameter of the piston is defined as a function of its displacement x (10):

$$D = D_0 \left(\frac{\tanh(k_1 x)}{k_2} + 1 \right) \quad (10)$$

For the present case the piston diameter varies following the curve shown in Figure 3. Parameters k_1 and k_2 are used for tuning the shape of the curve.

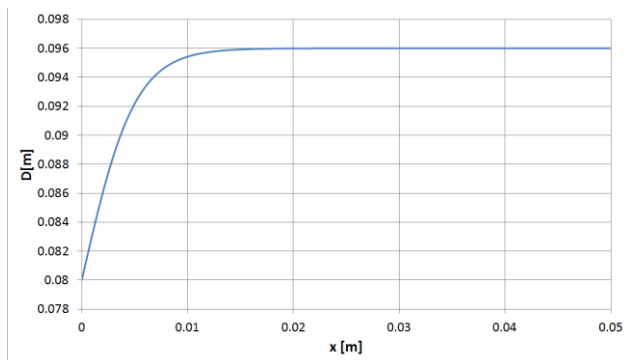


Figure 3: Piston diameter as a function of piston displacement

The volume of the system is defined as a function of the initial volume V_0 , the piston area A , and displacement x (11):

$$V = V_0 - \frac{Ax}{2} \quad (11)$$

Where the piston area A is defined as follows (12):

$$A = \pi \left(\frac{D}{2} \right)^2 \quad (12)$$

The pressure acting on the piston can be defined based on the ideal gas law (13):

$$p = \frac{nRT}{V} \quad (13)$$

Where n is the number of moles of gas present in the chamber of the air-spring and can be determined as follows (14):

$$n = \frac{p_0 V_0}{RT} \quad (14)$$

In the above equation (14) p_0 denotes the initial pressure of the air-spring system. For a displacement of 0.05 m the pressure evolution of the air-spring is presented in Figure 4 below.

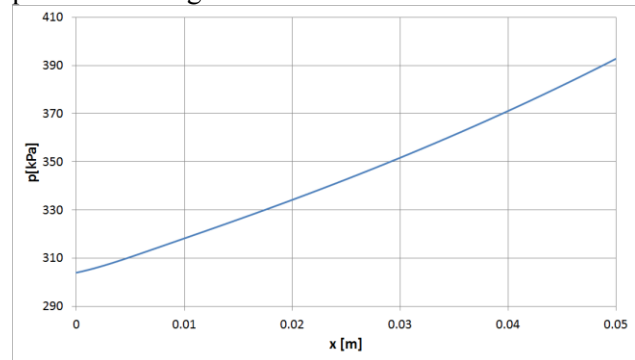


Figure 4: Pressure of the system as a function of piston displacement

The force acting on the piston is defined as a function of the piston area and the pressure in the air-spring system (15):

$$F = pA \quad (15)$$

Considering a displacement of 0.05 m, the evolution of the force acting on the piston is presented in Figure 5.

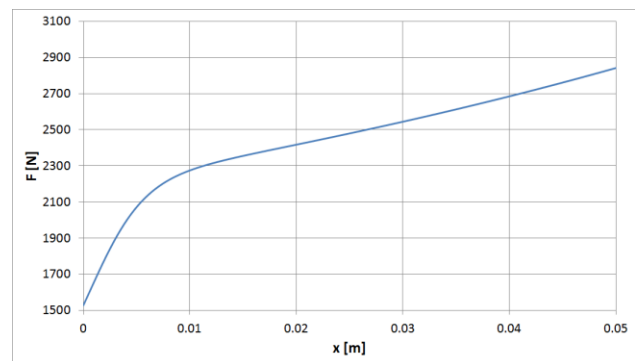


Figure 5: Piston force as a function of piston displacement

Based on the thermodynamic relations described above, the air-spring system was translated into Modelica code.

```

class Airspring
  parameter Real R = 8.3144621;
  parameter Real V0 = 0.0008;
  parameter Real T = 293.15;
  parameter Real p0 = 303975.0;
  parameter Real D0 = 0.08;
  parameter Real k1 = 200.0;
  parameter Real k2 = 5.0;
  Real V;
  Real n;
  Real A;
  Real p;
  Real D;
  input Real x;
  output Real F;
equation
  D = (D0 * tanh(k1 * x)) / k2 + D0;
  A = 3.14 * (D / 2.0) ^ 2.0;
  p = (n * (R * T)) / V;
  n = (p0 * V0) / (T * R);
  V = V0 + ((-A) * x) / 2.0;
  F = p * A;
end Airspring;
    
```

The pre-defined parameters of the Modelica code of the air-spring are the following:

$R = 8.3144621 [J/mol K]$	ideal gas constant
$V0 = 0.0008 [m^3]$	initial chamber volume
$T = 293.15 [K]$	gas temperature
$p0 = 303975 [Pa]$	initial gas pressure
$D0 = 0.08 [m]$	initial piston diameter
$k1 = 200$	parameter 1
$k2 = 5$	parameter 2

The input to the Modelica air-spring model is the displacement of the piston x and the output of the model is the force F acting on the piston.

An FMU for Model Exchange of the Modelica air-spring was generated with the specified IN and OUT ports, using OpenModelica 1.8.0 based on the FMI standard V1.0. This FMU was linked into a dynamic simulation with LMS Virtual.Lab Motion.

3.2 LMS Virtual.Lab Motion vehicle dynamics simulation with a Modelica air-spring FMU

In LMS Virtual.Lab Motion a front suspension of a vehicle was modeled (as shown in Figure 6). An Opposite Wheel travel scenario was implemented, which is one of the typical scenarios considered in vehicle suspension design for analyzing relevant suspension parameters and forces in the connecting elements.

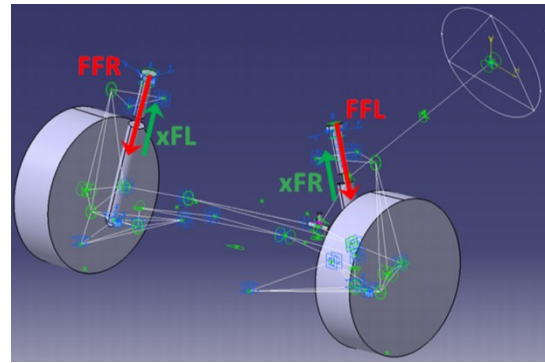


Figure 6: Vehicle front suspension in LMS Virtual.Lab Motion (air-spring FMU inputs are highlighted in green and outputs in red)

In an opposite wheel-travel analysis the left and right wheels are moved vertically on an equal but opposite path to simulate body roll. The left and right wheels move 180° out of phase with respect to each other along a specified bounce and rebound travel. For the present case, the wheel travel distance of 0.05m was considered with a cycle time of 1 s.

Two instances of the Modelica Air-spring FMU for Model Exchange were linked into the LMS Virtual.Lab Motion suspension model for the left and right side. The air-spring FMUs were linked to the upper and lower part of the damper units on the left and right side of the suspension.

Corresponding to the Modelica air-spring model the input to the air-spring FMU was the relative displacement of the lower damper part with respect to the upper part. In Figure 6, highlighted with green, xFL and xFR represent the relative displacement of the Front Left and Front Right dampers respectively.

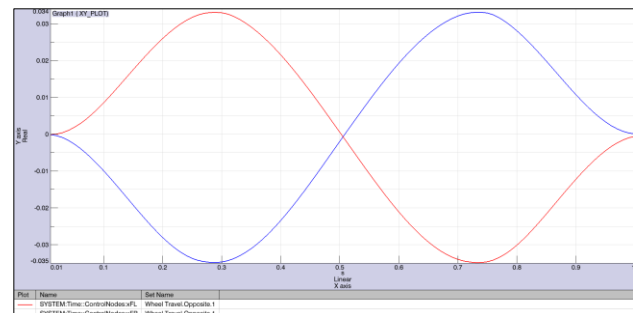


Figure 7: Air-spring FMU input signals (xFL in red and xFR in blue)

The evolutions of the FMU input signals for the left and right air-springs are presented in Figure 7.

The output of the FMU air-spring was the force on the piston of the air-spring, applied between the upper and lower damper part. Highlighted in red in Figure 6, for the left and right air-springs are the FMU output forces denoted with FFL and FFR respectively.

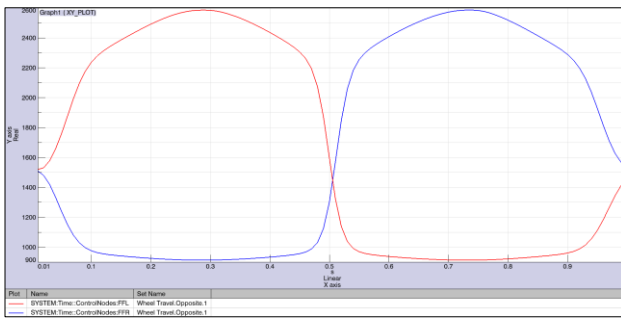


Figure 8: Air-spring FMU output signals (*FFL* in red and *FFR* in blue)

Figure 8 presents the evolutions of the FMU output signals. The nonlinear behavior of the air-spring forces is clearly visible.

3.3 Validation of the presented air-spring FMU with LMS Imagine.Lab AMESim

To validate the FMI implementation in LMS Virtual.Lab Motion, the results obtained with the FMU for Model Exchange have been compared to the results obtained with LMS Imagine.Lab AMESim.

LMS Imagine.Lab AMESim is a 1D simulation suite to model and analyze multi-domain, intelligent systems and predict their multi-disciplinary performance [8].

For the purpose of validation, the air-spring model has been replicated in LMS Imagine.Lab AMESim using the same equations (10–15). The AMESim model of the air-spring has been coupled with the LMS Virtual.Lab Motion model using a Model exchange approach, but instead of using the FMI standard, an internally developed interface was adopted.

Consequently, the set of control forces from the LMS Imagine.Lab AMESim air-spring have been applied on the LMS Virtual.Lab Motion mechanism, which have been solved together by the Virtual.Lab Motion solver. To be able to correctly compare results, the same communication time interval of $0.001s$ has been used for both cases.

Figure 9 presents the comparison of the different air-spring forces obtained with the FMU for Model Exchange with the LMS Imagine.Lab AMESim model. In this figure the front left air-spring force (*FFL*) is presented in red and the front right air-spring force (*FFR*) in blue. The FMU forces are depicted with continuous lines while the LMS Imagine.Lab AMESim forces are presented with dashed lines.

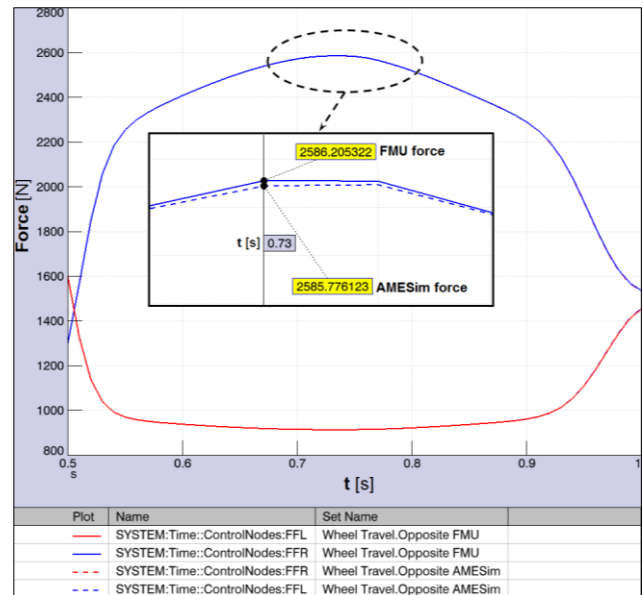


Figure 9: Comparison of Air-spring forces: *FFL* in red and *FFR* in blue; FMU signal in continuous line, AMESim signal in dashed line

As it can be noticed in Figure 9 the FMU forces and the AMESim forces follow very closely each other. In the central region of the figure, a close-up is presented at $t=0.73 s$.

The difference between the signals is $0.429 N$, which expressed in percentage, is approximately 0.016% and as such can be considered negligible.

4 Conclusions

The Modelisar FMI standard provides a vendor-neutral interface that allows the exchange of simulation models between different tools and platforms and enables their use in multidisciplinary simulations.

This paper presents the implementation of the Modelisar Functional Mock-up Interface (FMI) in LMS Virtual.Lab Motion. This functionality is demonstrated with an Opposite Wheel Travel scenario using a half vehicle model in LMS Virtual.Lab Motion and an Air-spring FMU for Model Exchange compiled from Modelica code.

Linking together different FMUs and an LMS Virtual.Lab Motion model in a co-simulation environment brings several benefits. However, both Co-simulation and Model Exchange type of simulation have their benefits and drawbacks.

In a Model Exchange type of simulation, in addition to the set of multibody equations of motion, a set of control forces from the FMU are applied on the mechanism, which are solved together by the Virtual.Lab Motion solver. Usually, the FMU forces

are the product of state equations. In a Model Exchange type of simulation the main benefits are: good numerical stability and use of the full capability of the solver (variable step sizes, iterative methods...). The drawback is that this approach may be inefficient and time consuming if large differences in stiffness exist between the subsystems and the systems are loosely coupled.

In case of Co-simulation, the coupling data is exchanged between the Virtual.Lab Motion solver and the FMU at each communication interval, consequently, the co-simulation approach is less stable. In the case of Co-simulation, the main benefits are: problem-specific solvers can be used for integrating different subsystems and hence it may be more time efficient for loosely coupled systems (solvers may use different integration step sizes). On the downside, this approach is less stable as the Model Exchange type. The main reason for this instability is the approximation of the coupling variables between two consecutive communication time steps. However, by choosing the communication step size carefully a stable simulation can be achieved.

As a result it is suggested to use the model exchange approach for tightly coupled systems, while the co-simulation approach may be more efficient in loosely coupled problems.

Acknowledgements

We gratefully acknowledge IWT Vlaanderen and ITEA2 for their support of the R&D projects IWT-080067 (ITEA2-07006) "MODELISAR" (From System Modeling to S/W running on the Vehicle), and we furthermore acknowledge IWT Vlaanderen for supporting the R&D project IWT-090408 "CHASING". In addition, we gratefully acknowledge the European Commission for their support of the Marie Curie IAPP project 285808 "INTERACTIVE" (Innovative Concept Modelling Techniques for Multi-Attribute Optimization of Active Vehicles, <http://www.fp7interactive.eu/>).

References

- [1] Anthonis J., Gubitosa M., Donders S., Gallo M., Mas P., Van der Auweraer H. Multi-Disciplinary Optimization of an Active Suspension System in the Vehicle Concept Design Stage, 14th Belgian-French-German Conference on Optimization, Leuven, Belgium, September, 2009.

- [2] Busch M. and Schweizer B. Numerical Stability and Accuracy of Different Co-Simulation Techniques: Analytical Investigations Based on a 2-DOF Test Model, 1st Joint International Conference on Multibody System Dynamics, Lappeenranta, Finland, May 25–27, 2010.
- [3] Schierz, T., Arnold, M. MODELISAR: Innovative numerische Methoden bei der Kopplung von multidisziplinären Simulationsprogrammen, in A. Brenke (ed.): Tagungsband ASIM-Konferenz STS/GMMS 2011, Krefeld, 24.02.-25.02.11, Shaker Aachen, 2011.
- [4] Functional Mock-up Interface: <http://www.functional-mockup-interface.org/>
- [5] IWT Vlaanderen, IWT-080067 (ITEA 07006) "MODELISAR" (From System Modeling to S/W running on the Vehicle), <http://www.modelisar.com>
- [6] Bastian J., Clauß C., Wolf S., Schneider P. Master for Co-Simulation Using FMI, Proceedings 8th Modelica Conference, Dresden, Germany, March 20-22, 2011.
- [7] LMS International, LMS Virtual.Lab Rev. 11, <http://www.lmsintl.com/virtuallab>, May 2011.
- [8] LMS International, LMS Imagine.Lab AMESim Rev. 11 <http://www.lmsintl.com/LMS-Imagine-Lab-AMESim>, May 2011.

Generating Functional Mockup Units from Software Specifications

Uwe Pohlmann,
Wilhelm Schäfer

Software Engineering Group,
Heinz Nixdorf Institute,
University of Paderborn, Germany
[upohl|wilhelm]@upb.de

Hendrik Reddehase, Jens Röckemann,
Robert Wagner

Solunar GmbH,
Gütersloh, Germany
[reddehase|roeckemann]@solunar.de
wagner@solunar.de

Abstract

This paper presents an approach to use the Functional Mockup Interface (FMI) for integration of classical controller specifications and statechart-based specifications of real-time critical message exchange protocols. The Functional Mockup Unit (FMU) is automatically generated from the specification. Using the generated FMU we are able to exploit simulation facilities as provided by Modelica/Dymola.

Keywords: Systems Engineering, Software Engineering, MechatronicUML, FMI, FMU, Modelica

1 Introduction

In today's globalized world market forces demand products to provide for more and more unique features. In so-called mechatronic or embedded systems these features are often realized (mainly) by software. For example, many new features which were recently introduced in the automotive industry are largely software driven.

In addition, very advanced new features will depend on extensive communication between currently still independently operating individual components. For example, intelligent lighting systems in cars will combine information about the environment obtained from their own sensors with those collected by other cars to save energy but also to avoid glaring other drivers. Similar examples exist for transportation systems in general but also for household appliances or in the production industry [25]. Here, possible significant energy savings are one main motivation to introduce so-called smart grids.

The resulting high amount of software enabling

communication between a large number of components combined with the software controlling individual components makes those systems more complex than today. This requires significant changes in the way software is developed today. This is especially true as the software controlling individual components is usually dealing with continuous variable values and developed by control engineers whereas software controlling communication is handling discrete input and output signals often using asynchronous communication and is developed by software engineers. In addition, electrical and mechanical engineers bring in expertise about the underlying hardware system constraints which have to be considered when developing the software.

As these systems are usually deployed in safety-critical environments, high quality of the software is an absolute must [21]. However, in the past, an overall validation of systems under construction was not possible until implementations had been finished, i.e., after all hardware and software parts had been built and integrated into the final product. The above mentioned different disciplines use their own models and formalisms to describe the corresponding parts of the system under development, e.g. feedback controllers are described using differential equations and communication protocols are described using statecharts. This development process hinders early (formal) verification and simulation of system models to detect errors in the design phase as early as possible and to avoid costly error removal in later development stages.

In this paper we focus on supporting simulation based on model-driven development especially considering cross-discipline development between control and software engineering. In contrast to other approaches like [22, 8, 26], we use

a discrete system model which enables the detailed specification of timing issues when specifying communication protocols, because message transfer specified by those protocols is real-time critical. Proper functioning of the system does not only depend on the correct order of messages sent and received but also on their timely delivery.

This paper presents how we employed the *Functional Mockup Interface* (FMI) and the *Functional Mockup Units* (FMU) in order to integrate discrete model-based real time protocol specification with controller design and appropriated simulation facilities using Modelica/Dymola.

The approach has been developed as part of the ENTIME project (ENTIME is the German acronym for 'Design Methods for Intelligent Mechatronics'). The project aims at the development of a seamless methodology reaching from conceptual design to concrete implementation of mechatronic systems. It is carried out in close cooperation with nine industrial partners. To support simulation of the physical models and corresponding feedback loops together with specifications of real-time protocols, the main challenge was to provide the needed tool support, because the project collaborators use different modeling and simulation tools in their industrial practice.

The paper is organized as follows. In the next section we illustrate the use of MechatronicUML, a domain specific modeling language enabling protocol specifications including sophisticated real-time constraints. The example which we use in the paper, is a miniature robot called BeBot which is a small mechatronic systems with a focus on ad-hoc communication. In Section 3, we give a brief and informal introduction to the concepts of the FMI standard, sketch our implementation of MechatronicUML according to the FMI standard for model exchange by means of the example, and present our tool support. Section 4 discusses related work in more detail. The paper closes with a conclusion and an outlook on future work.

2 Specification of Protocols

The specification language which we use is called MechatronicUML [3]. It has been developed by a large joint project between engineers and people from computer science. The project is the collaborative research center self-optimizing systems in mechanical engineering which is funded

by the German national science foundation since 2002 (<http://www.sfb614.de/en/>).

2.1 Running Example

The example is the scenario of a so-called obstacle avoidance maneuver which is performed by a BeBot. BeBot [11] is a sophisticated intelligent miniature robot, developed by the Heinz-Nixdorf Institute. Figure 1 shows a picture of a BeBot. In our scenario, the BeBot uses three sensors which detect obstacles in front, left and right of its current position. Further, the BeBot has a gyroscope which measures its current angle position with respect to the outside world. Three components of the BeBot are active when it performs obstacle avoidance. These components are (1) an exploration component which starts or stops the exploration of the environment, (2) a navigation component which steers the BeBot around an obstacle based on the given sensor inputs and (3) an obstacle detection component which receives the input from the three sensors and transforms them into corresponding messages which are received by the navigation component.



Figure 1: BeBot Robot [11]

As a consequence, the decision if and how an obstacle avoidance maneuver has to be performed depends on (extensive) asynchronous communication between these three components. For example, the navigator which knows the actual angle to the outside world, informs the obstacle detection component which sensor values are relevant. The obstacle detection component must not send messages when a turn is performed, because sensor values are not correct when the BeBot spins.

Figure 2 illustrates how a BeBot will find its way out of the shown maze.

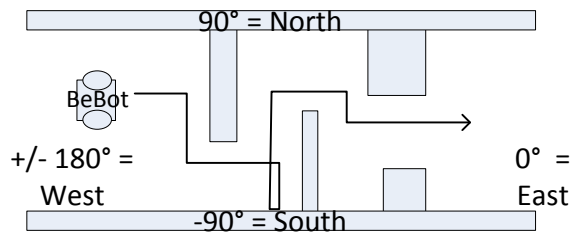


Figure 2: BeBot Obstacle Avoidance Maneuver

2.2 Structure Model

In MechatronicUML the system model is structured hierarchically and consists of either *atomic components* or of *structured components*. *Atomic components* implement their behavior directly and *structured components* are a composition of other components. The component model of MechatronicUML differs from other component-based approaches, like [27], as MechatronicUML employs active components, i.e. the behaviour of each component is specified by a real-time statechart (see below) and executed by a single thread [3].

Each component has interaction points, called *ports* for accessing their functionality. *Discrete ports*, shown as rectangles, are used for sending and receiving asynchronous messages. Each message is typed over a message type. Further, discrete message ports have the causality *in* \square , *out* \square , or *in/out* \square . Discrete in-ports can only receive messages, discrete out-ports can only send messages and in-out-ports can receive and send messages. A continuous port, shown as a triangle is either a continuous in-port \triangleleft , or a continuous out-port \triangleright . It sends or receives signal values which are typed as *Boolean*, *Int*, or *Real*.

Figure 3 shows the internal structure of the BeBot_SW component. It consists of three *atomic components*. The component Exploration is responsible for starting and stopping the exploration scenario. It is connected via its port *sender* to the component Navigation. The Navigation component is responsible for actuating the BeBot. It can set the *linear_speed* and the *angular_speed* of the BeBot. The Navigation component is connected to the ObstacleDetection component via its *discrete port* *master*. The ObstacleDetection component transfers the continuous signal values of the sensors *front*, *left*, and *right* to asynchronous messages. These messages inform the Navigation component if it has to perform an obstacle avoidance maneuver.

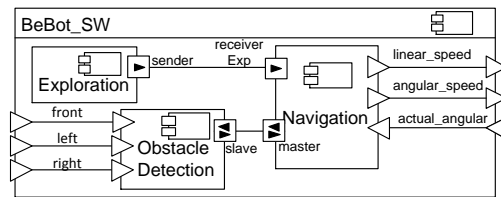


Figure 3: Component Type of the BeBot Software

2.3 Real-Time Properties

Real-Time properties are specified by clocks. In MechatronicUML a clock is a first-class real-valued entity and is used to synchronously measure the duration of time during execution. It can be reset to zero, which is marked by the keyword *reset*, with any state- or transition-action. At the beginning of the simulation clocks start with a zero-value. In contrast to *delayed transitions* of State Graph2 [22], or the *after*, *before*-construct of Stateflow [23], or the relative time event *after* of UML, a clock is not automatically reset when the system state changes. At any point in time, a clock can be read. The value of the clock represents the continuous-time since the last reset [2]. This semantics simplifies the specification of more complex real-time behavior and constraints. It is possible to compare clock values with time constants. We use clocks to specify transition guards, transition deadlines and time invariants of states.

2.4 Discrete Behavior Model

MechatronicUML uses Real-Time Statecharts to specify protocols of message exchange between different components, i.e. the order of message invocation and its corresponding time constraints. Besides elements from UML state machine formalism Real-Time Statecharts use syntactic elements like *clocks* and corresponding *clock constraints* as extended transition guards as defined by timed automata. In MechatronicUML each discrete port has its own statechart. The behavior of a component is given by the parallel composition of all statecharts of all its ports. In addition, it is possible to add synchronization channels like in timed automata to synchronize the behavior of the different port statecharts.

Time-invariants from timed automata constrain when and how long a statechart is allowed to stay in a particular state. We define the maximum time for evaluating and executing a transitions by a deadline. We use clocks as guards of

transitions, deadlines of transitions, and time invariants of states. The operational semantics of Real-Time Statecharts is formally defined by [12] and is based on timed automata.

It enables the application of formal verification techniques like real-time model checking [14] with tools like UPPAAL [4]. For instance we specify in our example in Figure 4 the safety property that each turn maneuver may not last longer than 5 seconds. Therefore we use the time invariant $c0 < 5$.

Figure 4 shows the Real-Time Statechart of the Navigation component. It consists of the parallel composition of the port statecharts receiverExp and master. The statechart in region receiverExp describes how the received messages from component Exploration are processed. At the beginning the statechart is in its initial state Stop and the parallel statechart master is in the state Halt. When the upper statechart gets the asynchronous message start the outgoing transition fires, if the synchronization channel go is available. The synchronization channel go is available if the sender transition, marked by the “!”, and the receiving transition, marked by the “?” can fire. If both transition can fire both transitions fire together in an atomic way. This means either both fire or none of them. Because there are no more conditions on the transitions they fire and the statechart gets in the states Start and Go.

When the statechart master enters the state Go the output signal linear_speed of the BeBot is set to the value 0.1 and the angular_speed is set to 0. In the state Go the BeBot drives forward until the ObstacleDetection sends the message obstacleFront. In this case the BeBot turns right to a southward direction and drives forward until the left sensor signals that there is no more obstacle at the left side. If there is no more obstacle, the BeBot turns back in an eastward direction and drives forward until the next obstacle occurs in front of it. If there is an obstacle at the left side until the BeBot reaches the corridor boarder, the BeBot performs a U-turn and drives forward until the right sensor signals that there is no more obstacle at the right side. These steps are carried out in a loop until the Exploration component sends the stop message.

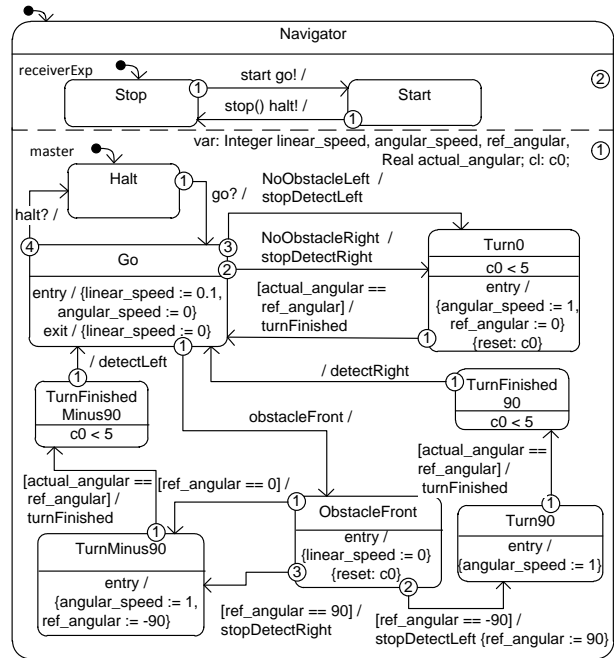


Figure 4: Real-Time Statechart the Navigator

2.5 Asynchronous Communication

The shown Real-Time Statechart formally defines the protocol definition of the message exchange and the corresponding timing constraints. Messages are sent when a transition fires. Messages which should be sent are shown behind the slash (/) and messages which should be consumed are shown before the slash. The connector may have a delay or a message could be lost. For the sake of simplicity of the figure above and due to lack of space, we omit the specification of the connector here. The receiver port of a message stores a received message in a mailbox. This is implemented as a queue and has a fixed size which is defined by the modeler during design time. Each message type has its own mailbox. Thus, the receiver can test directly if a needed message is available without searching the whole queue. Each message type could have an arbitrary number of parameters, which are packaged in the message when a transition fires. The receiver transition can read and process the parameters when it fires and consumes the message. Messages remain in the mailbox until a transition consumes and destroys it.

2.6 Further Features of MechatronicUML

As explained, MechatronicUML [3] mainly focuses on the discrete parts of systems. The language

especially addresses the specification of complex communication protocols with hard real-time requirements [9].

The structure of a mechatronic system is defined by a component-based development approach. It is possible to distinguish between discrete software components and continuous software components like controllers. MechatronicUML has clear interfaces between discrete system parts and continuous system parts.

The behavior of continuous components including their communication protocols is specified by an extension of our Real-Time Statecharts in the sense of hybrid automata. However, in contrast to hybrid automaton approaches [1, 19] we abstract from detailed definitions of controllers.

This abstraction together with some constraints on the parallel composition of port statecharts enables formal verification of the behavioral specification using model checking. We employ the model checker UPPAAL to verify safety properties like *deadlock freeness*, *state reachability* or *end-to-end response time*. MechatronicUML models can be verified automatically. We also prove by model checking that a mailbox will not overflow (see above). However, formal verification is beyond the scope of this paper and we refer to [15, 13] for further details.

3 Generating FMUs from Software Specification

This section shows how to generate an FMU.

3.1 FMI/FMU Fundamentals

Using different tools when designing the models leads to compatibility problems when you want to simulate all models in combination. To address this problem, the ITEA2 project MODELISAR has defined the FMI as an open standard for model exchange and co-simulation between multiple software systems. The FMI is used to create an instance of a model which can be loaded into any simulator providing an import function for FMI [7]. The *FMI for Co-Simulation* allows to couple several simulation tools [6].

A software instance compatible to the FMI is called an FMU. An FMU is basically a zip-archive with a “*.fmu” file extension. The information required for the simulation environment is collected

in an XML-file called *modelDescription.xml*. In addition, this file also includes a list of all variables available for data exchange between the simulator and the FMU. Furthermore, the standard defines functions that are used for the interaction between a model and the simulator. To provide an FMU, the FMU provider has to implement these functions using the C language.

3.2 Generating C-Code from MechatronicUML

This section sketches the C-code generation techniques for MechatronicUML models. The generated code may be used for a concrete microcontroller target platform or – as this paper shows – for an FMU implementation.

3.2.1 Generating C-code from the Structure Model

For each atomic component of the MechatronicUML model, we generate a header file and a corresponding implementation file. A component is mapped to a structure containing pointers to nested sub-components, variables, and clocks required for the associated statechart. In addition, corresponding code for the ports is generated. The discrete port implementation is used for inter-component communication. For this purpose, a discrete port implements an array of message queues. A queue stores messages of one specific type. For parametrized message types additional structures are generated in order to encapsulate the parameter values. For continuous in-ports we generated a variable with the causality input and for continuous out-ports we generate a variable with the causality output. The continuous port type is mapped to a corresponding FMI data type, e.g. *Boolean* to *fmiBoolean*. Via the input and output variables the FMU can be connected to other FMUs or Modelica/ Simulink components.

Our code is intended to run also on small 8-bit processors with only a few kilobytes of memory. This is too little to support both a real-time operating system and the control software. Hence, the control software is executed standalone on the processor and to support multiple communicating components on one processor, the components are processed in a cycle using a simple task loop implementation. Note that for future work we will intro-

duce a real-time operating system with more sophisticated task management and scheduling features for larger systems with 16- and 32-bit processors.

Listing 1 shows the execution sequence of our example. The information about a component is passed as an argument, allowing for multiple components of the same type to exist in one environment. In every processing cycle, a component statechart may exchange messages with other components by sending and receiving them. After every component has been processed, a synchronization step is performed where raised events are delivered to the target components.

```
...
// execute component behavior
exec_navigation(comp_navigation);
exec_exploration(comp_exploration);
exec_obstacle_detection(
    comp_obstacle_detection);
// execute message exchange
sync(connector_sender_receiverExp);
sync(connector_master_slave);
...
```

Listing 1: BeBot Execution Sequence

3.2.2 Generating C-Code from the Discrete Behavior Model

There are several implementation techniques for statecharts, but in most cases all the techniques are variants and combinations of (1) the state table, (2) the object-oriented state design pattern, and (3) the simple switch-case statement implementations. (1) The state table implementation maps directly to a state table representation in the code. As it is not hierarchical, it needs extensions for nested states and parallel regions and requires a large state table representation with a complicated initialization. Hence, the code is less readable. (2) The state design pattern simplifies the implementation of statecharts. However, it has also to be extended for hierarchical statechart implementations. In addition, the implementation is straightforward in C++, but it is rather complex in C, because of the needed mapping for inheritance and polymorphism. Therefore, we decided to generate nested switch-case statements (3). The implementation technique of switch-case statements is quite simple, it can be easily coded in C, and it has a small memory footprint since only one state variable is necessary to store the current state of a state machine. Furthermore,

nested switch-case statements allow us to implement hierarchical statecharts in a quite intuitive and readable way, which ensures traceability between the model and the generated code.

Listing 2 shows a code excerpt from the generated program for the BeBot example shown in Figure 4. It gives an impression of the generated C-code for a transition from state `ObstacleFront` to state `TurnMinus90`.

```
void execute_master(comp_navigation* comp) {
...
switch(reg_master) {
...
case STATE_NAVIGATOR_OBSTACLEFRONT:
if (ref_angular == 90) { ...
} else if (ref_angular == -90) { ...
} else if (ref_angular == 0) {
// state change
reg_master = STATE_NAVIGATOR_TURNMINUS90;
// entry actions
angular_speed = 1;
ref_angular = -90;
}
...
}
```

Listing 2: Excerpt from navigation.c

For each region of a statechart, we declare an Integer variable to keep the current state of this region. Within each case-statement, a sequence of mutually exclusive if-statements is used to determine whether one of the state's outgoing transitions can fire. In order to enable a transition, the existence of events and Boolean expressions generated from conditional guards, clock guards, and synchronization channels have to be evaluated. As transitions have priorities in MechatronicUML to prevent non-deterministic behavior, the generator sorts the transitions according to their priorities before generating the appropriate code. If a transition is enabled, it fires, i.e. the new state is set and the appropriate exit and entry actions are executed. A state may also contain do-actions and inner regions which are executed if no transition is enabled. For this case, a final else-block is created. Note, the presented program is executed once in each cycle. In order to use the generated C-code for an FMU implementation, we have to implement the required interfaces from the FMI standard.

3.3 FMI/FMU Wrap Up

We employ the FMU SDK from QTronic [16]. Figure 5 shows the relations and dependencies between the FMI standard, the QTronic FMU SDK, and our statechart implementation.

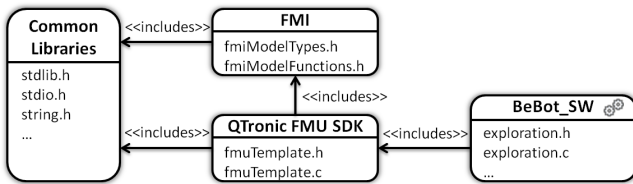


Figure 5: Implementation Dependencies

The basic implementation of the FMI is provided by the FMU SDK. To implement MechatronicUML, our code generator creates a header file and an implementation file for each component taking the FMU SDK into account.

The FMU SDK implements the FMI standard by delegating some of the tasks to supplementary functions that have to be implemented by the user. In our case, these implementations are also generated automatically by our code generator. The most important function is *eventUpdate*, as it is used to execute the statecharts. Since the *eventUpdate*-function is called by the FMI-function *fmiEventUpdate* whenever an event occurs during a simulation, we are able to react on changes in the simulation model. We use *Time Events* from the FMI to control the execution of a statechart at a regular interval and map each clock to an *fmiReal* variable. Since the current simulation time is passed to the FMU as a parameter, the current simulation time is assigned to the clock variable to reset a clock. Evaluating is done by calculating the elapsed time since the last reset. The difference between the current simulation time and the affected clock variable is used to evaluate clock constraints, deadlines and invariants upon appropriate actions are taken.

In the FMI standard, direct access to the data stored within the model is not possible, even if the source code is provided. Instead, a reference number is associated with each variable in the description file. Therefore, the FMU SDK stores variables of the model in four arrays of the types *fmiReal*, *fmiInteger*, *fmiBoolean*, and *fmiString* and references them by using indices. This is an efficient implementation of the FMI standard, but it is not useful for target-specific code that does not serve as an FMU implementation. Further, it is not easy to read and to understand the code. Therefore, we generate placeholders for the variables of our model. For the FMU implementation we generate preprocessor macros, which map the placeholders to FMU SDK compliant array access statements.

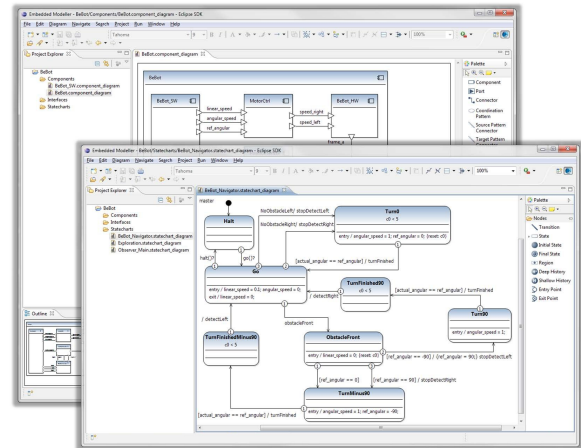


Figure 6: EMBEDDEDMODELLER

In case of other targets, e.g. microcontrollers, we generate preprocessor macros mapping the placeholder to more suitable structures and variables.

3.4 Tool Support

We provide our tool support in form of an Eclipse modeling tool suite which is called EMBEDDEDMODELLER. The EMBEDDEDMODELLER provides several diagram editors and supports software specifications based on MechatronicUML as explained in the previous sections. Figure 6 shows the editors for Real-Time Statecharts and structured component diagrams.

For generating C-code and the corresponding FMU description file, we used a template-based code generator framework. To create the FMU with all resources, the batch script provided with the FMU SDK is executed. Firstly, the batch file creates a temporary directory with the desired directory structure for the FMU under construction. Secondly, it compiles the sources and copies all needed files to the corresponding folders. Lastly, the batch packages the processed directory and saves it within the *.fmu file.

To simulate the designed BeBot software, we generated the FMU for our software specification BeBot_SW and created the Modelica model of the mechanical and control engineering parts of the BeBot within Dymola. The FMU was imported and connected to the hardware model of the BeBot. Since the continuous ports of the MechatronicUML serve as an interface to continuous components in general, we are able to connect our FMU, i.e. our discrete software component, to the provided BeBot feedback controllers. We simu-

lated the integrated model in Dymola successfully. Note, our approach is not limited to Modelica / Dymola as the FMI standard is tool-independent. Therefore, it is possible to simulate software specifications with any other simulation tool which supports the FMI model exchange standard.

4 Related Work

This section presents related work. We focus on approaches which can be used to simulate hybrid systems and where the discrete behavior is state-based. Further, we currently use the FMI for model exchange and not for co-simulation. Therefore, we do not discuss other approaches for co-simulation or distributed simulation like CODIS [5], TISC [20] or FMI co-simulation [6].

4.1 Statecharts in Modelica

Currently, state-based behavior can be modeled in Modelica with the library State Graph2 or algorithmic code is generated from SimulationX or ModelicaML.

State Graph2 is a Modelica library [22] which provides classes for states (*Step*), hierarchical and parallel behavior (*Parallel*), and transitions (*Transition*). With these elements it is possible to model complex behavior like Harel's wrist watch example. In contrast to MechatronicUML, StateGraph2 has no concepts for clocks, clock constraints, time invariants, and deadlines to specify and constrain timing behavior. A modeler could manually implement such behavior in Modelica. Further, State Graph2 has no concept for asynchronous message-based communication. We are currently working on such a library extension. However, as State Graph2 is modeled with equations and these equations are sorted before the model is simulated, the modeler can hardly influence the resulting C-code generated by Dymola. So, it is difficult to compare this code with real target source code. The FMI C-code is the same as the target source code except for the interface definition .

SimulationX has its own state-based language which follows the ideas of UML state machines and supports a subset [8]. In contrast to Real-Time Statecharts, SimulationX does not support parallel behavior, timing behavior, and coordination of distributed components by asynchronous

communication. Timing behavior is supported in a limited way, as transition firing could be constrained to a time interval from the moment when the source state of the transition is entered. In MechatronicUML we use, like timed automata, clocks, clock constraints, time invariants, and deadlines to specify and constrain the timing behavior of our models. Messages are only available within a statechart in SimulationX. They do not support an arbitrary number of parameters, and messages are lost when a transition cannot react on the event immediately. Therefore, it is difficult to specify coordination of distributed components. SimulationX generates Modelica algorithm code from its state machines.

ModelicaML is a UML Profile [26] which enables to use UML *Classes* and *Properties* to specify Modelica models. State-based behavior is modeled by UML state machines. The code generation mechanism supports nearly all UML state machine constructs [24]. The Modelica code is generated like SimulationX to the Modelica algorithm section. As UML has no concept for clocks, clock constraints, time invariants, and deadlines, ModelicaML does not support them either. Asynchronous messages between components can be simulated via an external C-function [24]. As ModelicaML has all freedoms of Modelica, it is not possible to verify the resulting models efficiently.

4.2 MATLAB Simulink/Stateflow

MATLAB has an own state-based modeling language called Stateflow, which can be combined with its simulation platform Simulink. Stateflow supports many features from UML state machines and can be combined with the whole capabilities of the MATLAB platform via its action language. It is only possible to define formal semantics for restricted Stateflow models [17]. Stateflow does not provide first class modeling entities for specifying timing behavior, except simple *after* and *before* statements. Stateflow does not provide a concept of buffering messages. It is possible to model such elements with a combination of Simulink and Stateflow blocks, but this is complex, error prone, and hard to maintain manually [18]. It is possible to load FMU using the separate FMI toolbox of Modelon [10].

5 Conclusion and Outlook

This paper shows how it is possible to generate FMUs from a formal software specification language for cyber-physical system. As a result it is possible to perform software-in-the-loop tests by numerical simulation of hybrid systems. We describe the following problem that arises when providing a methodology and tool support reaching from conceptual design to concrete implementation of cyber-physical systems: The approach should support the overall system simulation for different industrial partners in a heterogeneous design tool environment. The partners provide simulation models for mechanical and control engineering parts of the system, but software simulation models are missing. The transformation of software specification to FMUs solves this problem.

As the main contribution, we describe how a software specification in MechatronicUML can be automatically translated to FMUs maintain the original MechatronicUML semantics and, thus, the verification results. In particular, we map the component-based structure, the asynchronous communication in form of Real-Time Statechart, and real-time properties in MechatronicUML to C-code, which is wrapped by the FMI. We implemented the generation of FMUs from a given MechatronicUML model using a model-driven transformation approach. This combines the modeling and formal verification strengths of MechatronicUML with the advanced simulation capabilities of simulation tools like Dymola or Simulink. As a result of numerical errors we cannot guarantee that in different FMI import tools the different simulation runs have the same behavior. Therefore, the formal verification is important because it proves every possible simulation run and guarantees that all paths are conform to the specification. It is up to further research to proof that our generation is correct and keeps the verified properties.

The shown transformation approach should be interesting for anyone who wants to test formal software specification by simulation against a model of the physical system. A transformation against the FMIs could be performed for other formal software specification languages like Petri nets for flow analysis or stochastic software models for testing performance or failure rates. Hereby, it would be possible to combine the strength of formal analysis and numeric simulation.

For future work, we plan to develop a concept to allow for communicating via messages between several FMUs. Further, we want to generate code against different hardware platforms to analyze the timing behavior. We want to integrate the behavior of an underlying middleware or real-time operating system into the simulation. We may use co-simulation for this purpose. The simulation of complex cyber-physical systems requires much computing time. We want to compare the performance of native Modelica simulations with integrated FMU simulations and try to enhance the performance of hybrid simulations. Currently, it is not easy to interpret the simulation results. Here a bisimulation concept would help. To show the result, a simulation run could be visualized in the statechart or the message exchange could be visualized by sequence diagrams.

Acknowledgments

This work was developed in the project 'ENTIME: Entwurfstechnik Intelligente Mechatronik' (Design Methods for Intelligent Mechatronic Systems). The project ENTIME is funded by the state of North Rhine-Westphalia (NRW), Germany and the EUROPEAN UNION, European Regional Development Fund, 'Investing in your future'.

References

- [1] 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.
- [2] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.
- [3] S. Becker, C. Brenner, S. Dziwok, T. Gewing, C. Heinzemann, U. Pohlmann, C. Priesterjahn, W. Schäfer, J. Suck, O. Sudmann, and M. Tichy. The mechatronicuml method - process, syntax, and semantics. Technical Report tr-ri-12-318, Software Engineering Group, Heinz Nixdorf Institute University of Paderborn, 2012.
- [4] G. Behrmann, A. David, and K. Larsen. A tutorial on uppaal. *Formal methods for the design of real-time systems*, pages 33–35, 2004.
- [5] F. Bouchhima, G. Nicolescu, E. M. Aboulhamid, and M. Abid. Generic discrete-continuous simulation model for accurate validation in heterogeneous systems design. *Microelectron. J.*, 38(6-7):805–815, June 2007.

- [6] MODELISAR Consortium. Functional mock-up interface for co-simulation. version 1.0, 2010. www.functional-mockup-interface.org.
- [7] MODELISAR Consortium. Functional mock-up interface for model exchange. version 1.0, 2010. www.functional-mockup-interface.org.
- [8] U. Donath, J. Haufe, T. Blochwitz, and T. Neidhold. A new approach for modeling and verification of discrete control components within a Modelica environment. In *Proceedings of the 6th Modelica Conference (Modelica 2008)*, Bielefeld, pages 269–276, 2008.
- [9] S. Dziwok, C. Heinzemann, and M. Tichy. Real-time coordination patterns for advanced mechatronic systems. In *Proceedings of the 14th International Conference on Coordination Languages and Models (COORDINATION 2012)*, pages 166–180, June 2012.
- [10] S. Gaaloul, B. Delinchant, F. Wurtz1, and F. Verdière. Software components for dynamic building simulation. In *Proceedings of Building Simulation 2011: 12th Conference of International Building Performance Simulation Association, Sydney, Australia*, pages 2278–2284, November 2011.
- [11] J. Gausemeier, T. Schierbaum, R. Dumitrescu, S. Herbrechtsmeier, and A. Jungmann. Miniature robot bebop: Mechatronic test platform for self-x properties. In *Proceedings of the 9th IEEE International Conference on Industrial Informatics (INDIN 2011)*, pages 451–456, July 2011.
- [12] H. Giese and S. Burmester. Real-time state-chart semantics. Technical Report tr-ri-03-239, Lehrstuhl für Softwaretechnik, University Paderborn, Paderborn, Germany, June 2003.
- [13] H. Giese, S. Burmester, W. Schäfer, and O. Oberschelp. Modular design and verification of component-based mechatronic systems with online-reconfiguration. In *Proceedings of 12th ACM SIGSOFT FSE*, pages 179–188, 2004.
- [14] H. Giese, M. Tichy, S. Burmester, W. Schäfer, and S. Flake. Towards the compositional verification of real-time uml designs. In *Proc. of the 9th European software engineering conference held jointly with 11th ACM SIGSOFT international symposium on Foundations of software engineering (ESEC/FSE-11)*, 2003.
- [15] H. Giese, M. Tichy, S. Burmester, W. Schäfer, and S. Flake. Towards the compositional verification of real-time uml designs. In *Proceedings of 9th ESEC and 11th ACM SIGSOFT FSE*, pages 38–47. ACM Press, 2003.
- [16] QTronic GmbH. FMU SDK 1.0.2, 2011. www.qtronic.de/de/fmusdk.html.
- [17] G. Hamon and J. Rushby. An operational semantics for Stateflow. *International Journal on Software Tools for Technology Transfer (STTT)*, 9(5):447–456, 2007.
- [18] C. Heinzemann, U. Pohlmann, J. Rieke, W. Schäfer, O. Sudmann, and M. Tichy. Generating simulink and stateflow models from software specifications. In *Proceedings of the International Design Conference (DESIGN 2012) Dubrovnik, Croatia*, May 2012.
- [19] T.A. Henzinger. The theory of hybrid automata. In *Logic in Computer Science, 1996. LICS'96. Proceedings., Eleventh Annual IEEE Symposium on*, pages 278–292. IEEE, 1996.
- [20] R. Kossel, W. Tegethoff, M. Bodmann, and N. Lemke. Simulation of complex systems using modelica and tool coupling. In *Proceedings of the 5th International Modelica Conference (Modelica 2006)*, pages 485–490, 2006.
- [21] P. Marwedel. Embedded and cyber-physical systems in a nutshell. *DAC. COM Knowledge Center Article*, 2010.
- [22] M. Otter, M. Malmheden, H. Elmqvist, S.E. Mattsson, C. Johnsson, D. Systèmes, and S.D. Lund. A new formalism for modeling of reactive and hybrid systems. In *Proceedings of the 7th Modelica Conference (Modelica 2009)*, Como, Italy, pages 364–377, 2009.
- [23] M. Pajic, Z. Jiang, I. Lee, O. Sokolsky, and R. Mangharam. From verification to implementation: A model translation tool and a pacemaker case study. In *Proceedings of the 18th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2012)*, Beijing, China, April 2012.
- [24] U. Pohlmann and M. Tichy. Modelica code generation from modelicaml state machines extended by asynchronous communication. In *Proceedings of the 4th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools, EOOLT 2011, Zurich, Switzerland*, 2011.
- [25] W. Schäfer and H. Wehrheim. Model-driven development with mechatronic uml. *Graph transformations and model-driven engineering*, pages 533–554, 2010.
- [26] W. Schamai. Modelica modeling language (modelicaml) : A uml profile for modelica. Technical report, Linköping University, Department of Computer and Information Science, The Institute of Technology, 2009.
- [27] C. Szyperski. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley, 1998.

Functional Mock-up Interface in Mechatronic Gearshift Simulation for Commercial Vehicles

Andreas Abel² Torsten Blochwitz² Alexander Eichberger³

Peter Hamann¹ Udo Rein¹

¹Daimler AG, HPC B209, 70546 Stuttgart, Germany

²ITI GmbH, Webergasse 1, 01067 Dresden, Germany

³SIMPACK AG, Friedrichshafener Str. 1, 82205 Gilching, Germany

Abel@ititim.com, Blochwitz@ititim.com, Alex.Eichberger@SIMPACK.de,

Peter.Hamann@daimler.com, Udo.Rein@daimler.com

Abstract

Mechatronic shifting simulation of automated transmissions in commercial vehicles is used for optimization and development in today's truck engineering departments at Daimler. Within the ITEA2 project Modelisar in cooperation with ITI GmbH and SIMPACK AG this application served as a usecase for proof of concept of the newly developed Functional Mock-Up interfaces (FMI). Utilizing these standardized interfaces models from different tools are coupled to build up the overall system for the mechatronic shifting simulation. The coupling via FMI for Model Exchange was achieved for control modules from MATLAB/Simulink into the SimulationX powertrain model and secondly from the 1D-multiphysics powertrain in SimulationX into multibody vehicle in SIMPACK. Furthermore FMI for Co-Simulation was investigated in a pure SimulationX framework for the powertrain model. Very promising results can be observed as for modeling as for simulation processes. The FMI technology has clearly shown its capability to be applied in the productive simulation process.

Keywords: FMI, Modelisar, multibody system, automated shifting, mechatronics, co-simulation, model exchange

1 Introduction

The ITEA2 project Modelisar was a European research initiative from 2008 till 2011 focusing on the overall development process "from System Modeling to S/W running on the Vehicle". The major outcome is the standardization proposal Functional Mock-Up Interface (FMI) to facilitate tool and model coupling on implementation and numerical level, e.g. see [4], [5]. Within the project several usecases served as proof of concept by utilizing tools providing the new interfaces.

One usecase was the "Mechatronic shifting simulation of commercial vehicles" provided by Daimler AG, ITI GmbH and SIMPACK AG. This workpackage uses a Functional Mock-Up (FMU) of an automated gear shift system within the truck powertrain focusing on the transmission and demonstrating several benefits of the Modelisar environment. The SiL simulation is used for optimizing gear shift times and shifting comfort in heavy-duty trucks. Major challenges are the number of sub-models from different simulation tools and the necessary standardization of modeling, coupling, and solving.

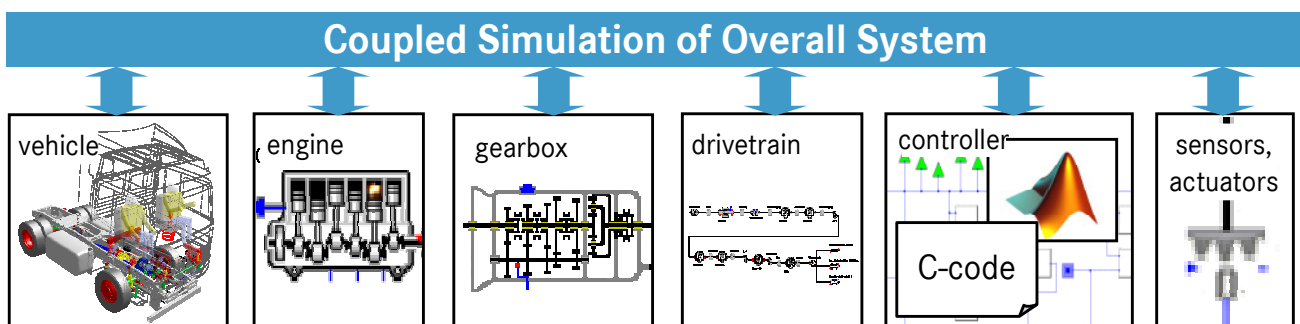


Fig 1.1 - FMU Mechatronic Shifting Simulation

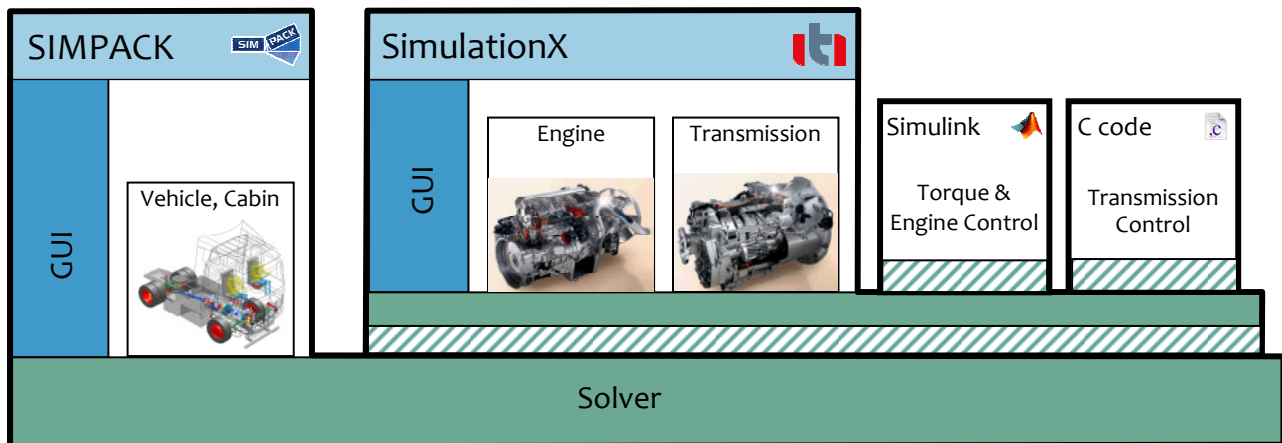


Fig 2.1 – FMI for Model Exchange Usecase Prototype

2 Mechatronic shifting simulation

2.1 The Modelisar usecase

The demonstration target for the Modelisar usecase has been a fully shiftable powertrain of heavy-duty trucks, which models the physics and control structure of the overall vehicle in such a degree of detail, that all phases of a gear shift can be reflected in terms of interactions between the driveline and vehicle dynamics, and the different control units participating in the shift. The models are capable to treat the large-scale low-frequency effects such as driveline jerking, as well as high-frequency phenomena, such as dynamics of actuation systems and gearbox components, engine combustion, or the impact of CAN bus delays on the overall system behavior.

The model allows the accurate prediction of the performance of the different driveline control units in interaction with the truck, the vehicle responses, and the perception of the driveline operation by the driver in terms of driving and shifting comfort.

This type of modeling requires the integration of a heterogeneous collection of models created for various simulation environments:

- 1D torsional vibration models of engine, gearbox – modeled in SimulationX^①,
- Actuation systems on clutch and gearbox – modeled in SimulationX,
- Detailed multibody (mbs) vehicle – modeled in SIMPACK^②

- Controller model from external supplier – compiled C code generated from MATLAB/Simulink
- DAIMLER in-house controller code – C code and MATLAB/Simulink models,
- Re-engineered controller functionality where no appropriate source was available – modeled in Modelica^③ within SimulationX.

Thus, this application has been a perfect target within the MODELISAR project to verify technologies developed in the project framework, for the FMI as well as for FMI-based co-simulation technologies.

The main objectives of the usecase were decreased simulation time, improved processes and overall decreased development times utilizing the FMI coupling techniques. Therefore the Simulation-in-the-Loop (SiL) implementations are based on the FMI-enhanced new versions of SimulationX, SIMPACK and MATLAB/Simulink representing control functionalities and powertrain models.

2.2 Simulation environment at project start

The usecase started with a simulation framework as shown in Figure 2.2, see [1], [2]. It included already all necessary models for the mechatronic shifting simulation. The coupling was based on proprietary non-standard interfaces from SimulationX and SIMPACK.

¹ SimulationX is a registered trademark of ITI GmbH

² SIMPACK is a registered trademark of SIMPACK AG

³ Modelica is a registered trademark of the Modelica Association

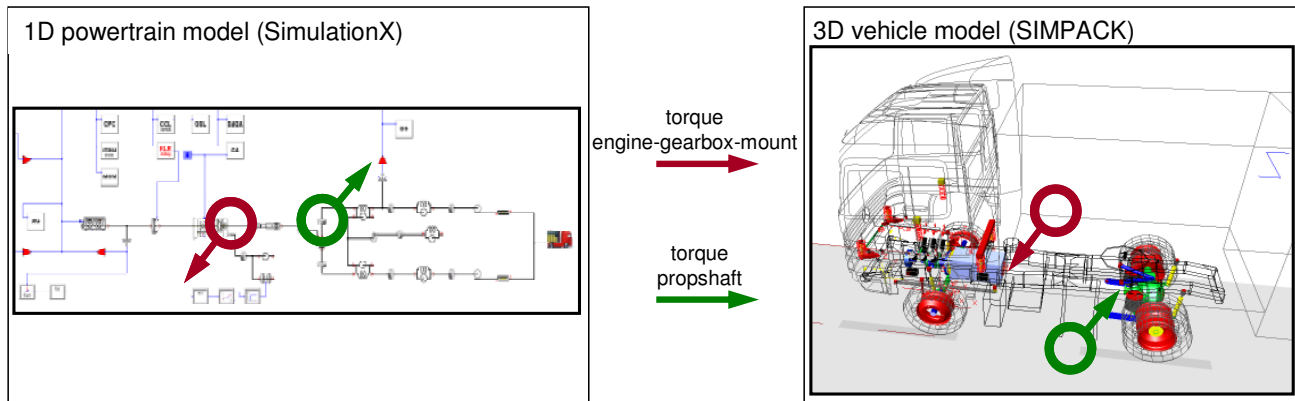


Fig 2.2 – Simulation environment at project start

The control unit was integrated into the SimulationX powertrain model utilizing Modelica external functions on one hand via MATLAB®/Simulink® and a Real-Time Workshop®⁴ (RTW) SimulationX target or the other hand via wrapped exported C-Code. The resulting time excitations stimulated the vehicle model in SIMPACK as an offline coupling.

Although this simulation environment already delivered detailed and qualitatively good results many reasons for improvement were observed: The offline coupling omits any feedback of powertrain and vehicle. The model exchange was as well for the software modules as for the powertrain module mainly handwritten and error prone, inefficient and costly to maintain.

3 FMI for Model Exchange

Within this use case the FMI for Model Exchange has been the main instrument to achieve the desired tool interoperability and model transfers. Using the FMI for Model Exchange controllers have been connected to a SimulationX driveline model, which in turn has been integrated into SIMPACK using the very same technology, see [3].

3.1 Control unit integration

Control software development for engine, clutch and gearbox control for Daimler commercial vehicles is done mainly inhouse. Software development stretches over a wide and dynamic range of MATLAB/Simulink and TargetLink®⁵ versions but also includes plain C code. Thus achieving a software-in-the-loop (SiL) integration of these control unit mod-

ules provides a certain challenge but has also a high potential to be facilitated with a standardized interface such as FMI. Within the usecase FMI applicability has been examined in two variants:

Variant 1 applies to controllers implemented in MATLAB/Simulink and utilizes a RTW toolbox provided by Dassault Systèmes AB through another Modelisar work package in order to export the Simulink model as FMU. Such FMU can be integrated easily into SimulationX. So this variant was the primary path in the usecase to verify the FMI interface.

Variant 2 implemented an FMI wrapper for plain controller C code. This was tested only prototypically within Modelisar using small test models. The approach has been demonstrated to be applicable but the absence of an automated FMI wrapper generation (e.g. through scripting) and the necessary degree of manual preparation of the FMU so far does not allow a productive use in the overall simulation process.

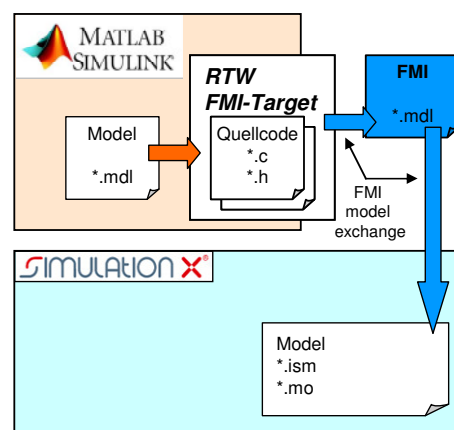


Fig 3.1 – Process chain for transferring MATLAB/Simulink controller models into SimulationX

The migration to FMI based SiL controller models has been achieved within the productive simulation process by gradually replacing the tool-specific solutions based on a dedicated SimulationX target for the Real-Time Workshop (RTW).

⁴ MATLAB, Simulink and Real-Time Workshop are registered trademarks of The MathWorks, Inc.

⁵ TargetLink is a trademark of dSPACE GmbH

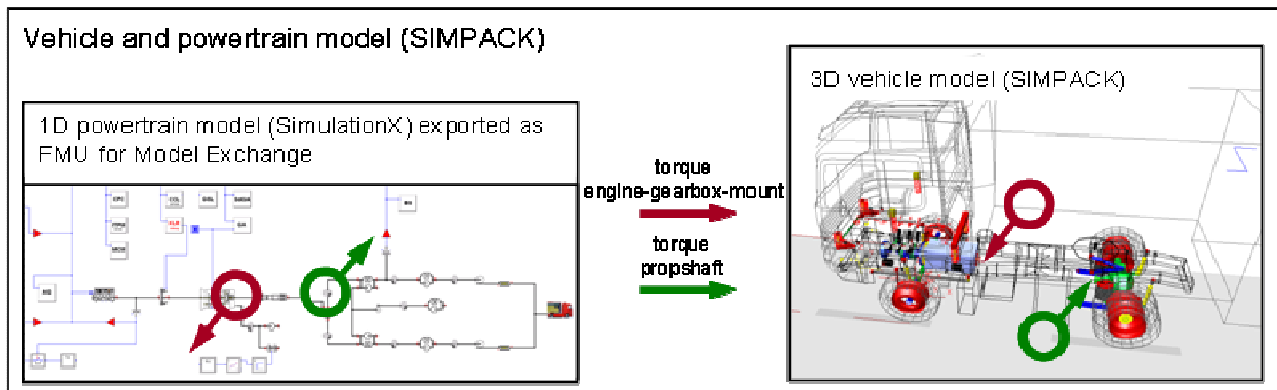


Fig 3.2 – FMI without feedback

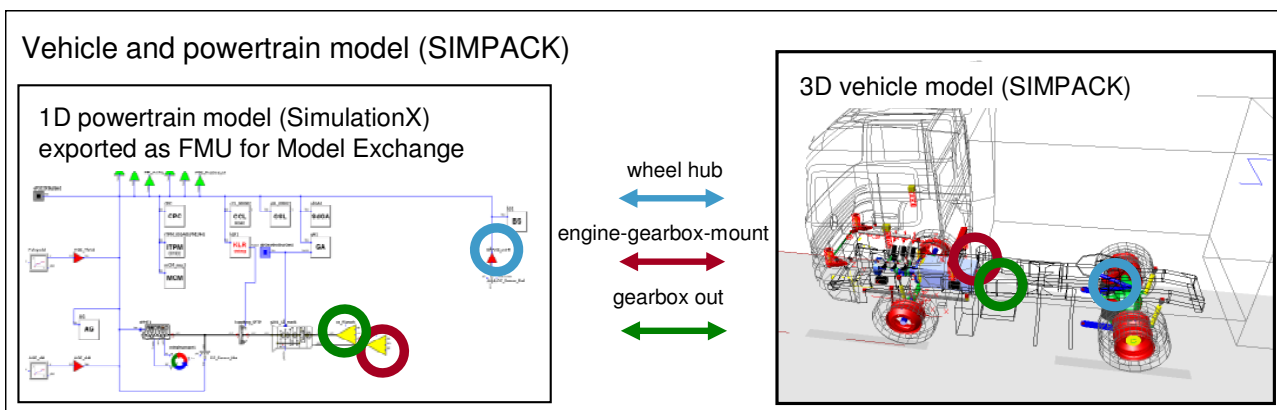


Fig 3.3 – FMI with feedback

Applying the standardized FMI technology now allows a higher degree of automation in the model exchange process and significantly broadens the range of potential target environments for the mechatronic gearshift simulation. The application of FMI also did not affect results and simulation performance.

3.2 FMU import without feedback

In a first stage prototype, the powertrain FMU including the control unit of section 3.1 was imported by FMI in the SIMPACK vehicle model. Rather than using offline pre-calculated inputs to the mbs-model as shown in section 2.2, online generated inputs were used in the simulation (see Figure 3.2).

Cutting point of the SIMPACK powertrain model is between the first cardan shaft and the differential on the rear axle. All rotational parts in front of the cardan joint are modeled as 1-D model within SimulationX and imported via FMI in SIMPACK. All rotational parts behind the cardan joint and the complex vehicle 3D-structure are modeled in SIMPACK.

No input from SIMPACK is passed into the FMU during simulation (without feedback).

FMU output, being applied to SIMPACK as kinetic excitations:

- torque on the gearbox output,
- torque on the rear differential input.

The results of the FMU integration without feedback show a perfect match with the results of the approach of section 2.2. The SIMPACK solver step size of the SIMPACK vehicle model without FMU was approximately 1e-3s. By integrating the FMU in the SIMPACK model, the overall simulation time increased due to the limiting step size of 20e-6s induced by the FMU. The integration time can potentially be reduced by a co-simulation between these two models, see section 4. Another approach is a performance optimization of the powertrain model in SimulationX. First model analyses show a high potential to at least gain an acceleration of factor 10.

3.3 FMU import with feedback – full FMI for Model Exchange solution

In the final stage prototype, the dynamic feedback between powertrain FMU and vehicle model was taken into account (see Figure 3.3)

Cutting point of the SIMPACK powertrain model is behind the gearbox output flange. All rotational parts in front of this flange are modeled as 1D model

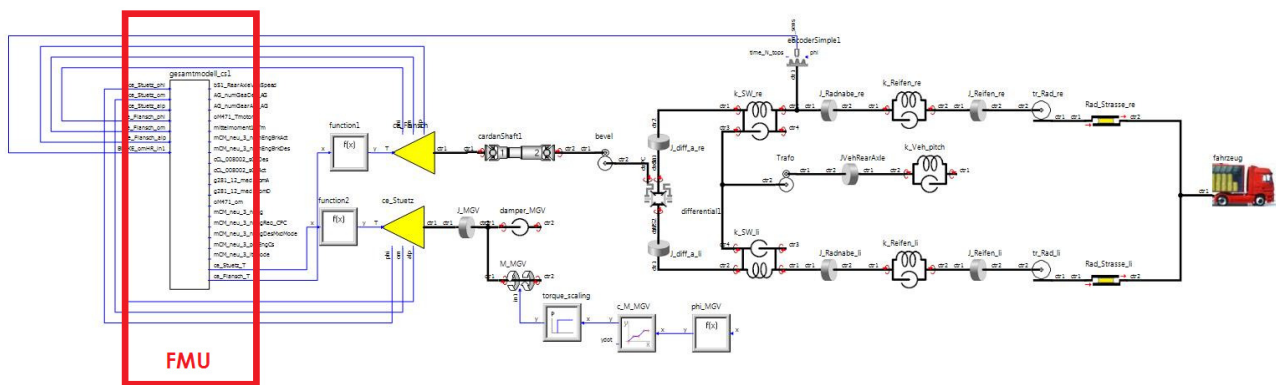


Fig 4.1 – FMI for Co-Simulation with Vehicle Model in SimulationX

within SimulationX and imported via FMI in SIMPACK. All rotational parts behind this flange and the complex vehicle 3D-structure are modeled in SIMPACK.

The FMU uses the following values as input (with feedback), being kinematic measurements of SIMPACK:

- relative angle of the engine block wrt. vehicle frame,
- relative rot. velocity of the engine block wrt. vehicle frame,
- relative angle of the front propeller shaft flange wrt. engine block,
- relative rot. velocity of the front propeller shaft flange wrt. engine block,
- angle of the rear wheels.

FMU output, being applied to SIMPACK as kinetic excitations:

- torque on the gearbox output,
- torque on the front propeller shaft flange.

The feedback introduces a new level of accuracy for simulation of shifting comfort and increases simulation quality at Daimler.

The simulation performance shows the same characteristics as described in chapter 3.2. The step size is dominated completely by the high dynamic powertrain model and no additional difficulties due to the feedback are introduced. The future work will focus on FMI for Co-Simulation on one hand and performance increase in the powertrain model on the other hand.

4 FMI for Co-Simulation

Sharing models between different simulation tools using the FMI for Model Exchange potentially provides pitfalls if the tools are using different solver technologies or if models run on significantly different time scales. The latter is also the case in the

model coupling between SimulationX and SIMPACK.

The FMI for Co-Simulation is a potential solution since it allows bundling a model with a dedicated solver, which can run independently of the solver in the target system. On the other hand a co-simulation between models of physically coupled sub-systems may be difficult due to the inherently introduced communication time delays in the coupling.

During the usecase project the FMI for Co-Simulation in SIMPACK still has been under development. For this reason an alternative test scenario has been implemented, where the FMI of the driveline which is intended for SIMPACK integration has been re-imported into SimulationX and has been coupled with a model of the downstream driveline (from the differential onwards), see [3]. In terms of present natural frequencies and discontinuities this reflects a similar scenario as in SIMPACK, where the vehicle part of the model could be solved with significantly bigger time steps.

The maximum achievable stepsize in the communication and the impact of different interpolation methods between communication steps have been assessed. This test case showed a clear need to balance the communication stepsize in order to achieve stable and valid simulation results. Although the communication for a physical link still needs very small step sizes, the communication stepsizes are about 10 times larger than the required stepsizes using an FMI for Model Exchange. This allows expecting an equivalent 10-fold performance increase.

5 Conclusions and Outlook

The usecase Mechatronic Shifting Simulation was completed successfully by implementing an FMI-based simulation framework already improving the development at Daimler and furthermore showing high potential for a wider future use. The new interface standardization proposals FMI for Model Ex-

change and FMI for Co-Simulation were implemented in different prototypes arising in the field of drivetrain dynamics and especially mechatronic shifting simulation for commercial vehicles.

To model and simulate the mechatronic powertrain SimulationX is used. Since version 3.4 SimulationX implements FMI for Model Exchange as import and export, as well as FMI for Co-Simulation as Slave and Master. All four variants have been tested successfully in different prototypes.

The vehicle and cabin is modeled as 3D multi-body system in SIMPACK. The version SIMPACK 9.0 implements FMI for Model Exchange as import. This interface has been tested successfully for different prototypical implementations.

Furthermore the control unit software had to be imported from MATLAB/Simulink. This could be achieved successfully via FMI for Model Exchange with the RTW toolbox as well developed within Modelisar.

From a technical point of view the following result could be achieved:

- Prototypical FMI based simulation of mechatronic shifting system with bi-directional coupling of vehicle and powertrain model,
- FMI based model exchange of MATLAB/Simulink control modules into powertrain model in SimulationX,
- FMI based model exchange of SimulationX powertrain model into SIMPACK vehicle model, but simulation performance needs further improvement due to problem specific multi-scale behavior,
- Alternative bi-directional coupling via FMI for Co-Simulation started.

FMI makes coupling of models easier to implement. The numerics of the coupling regarding performance, model harmonization, etc. must be analyzed as before. The potential of FMI for Co-Simulation could be shown. As soon as implementations are available it will be investigated for the coupling of vehicle and powertrain model for the mechatronic shifting simulation.

In the result of the Modelisar project the further development and improvement of the FMI standard has become a core task within the Modelica community. Due to the high industrial acceptance and feasibility proven by projects such as the presented use case, tool vendors eagerly follow these developments within their tools.

SIMPACK Version 9 officially supports Model Import based on FMI Standard 1.0. Co-Simulation based on FMI Standard 1.0 is currently under development and will be available fall 2012.

ITI has been driving FMI developments from the very beginning and fully supports all FMI variants in SimulationX since 2010.

As soon as FMI Standard 2.0 has been officially released, SIMPACK and ITI are going to upgrade the FMI interfaces in their tools to this version.

To summarize it can be said that with the new coupling interfaces enormous benefits for industrial applications can be generated: They reduce significantly the implementation complexity and costs for tool and model coupling. They optimize the SiL processes regarding time, cost, robustness and quality. They simplify internal and external model exchange and model reuse.

References

- [1] A. Abel (ITI GmbH), P. Hamann, U. Rein (Daimler AG) Modelisar: sWP303 Mechatronic shifting simulation, Milestone report: Documentation of specification and simulation environment, ITEA2 Modelisar internal paper, 2009
- [2] A. Abel (ITI GmbH), P. Hamann, U. Rein (Daimler AG) Modelisar: sWP303 Mechatronic shifting simulation, Milestone report: Results prototype 1, ITEA2 Modelisar internal paper, 2010
- [3] A. Abel, T. Blochwitz (ITI GmbH), M. Friedrich, J. Zeman (SIMPACK AG), P. Hamann, U. Rein (Daimler AG) Modelisar: sWP303 Mechatronic shifting simulation, Milestone report: Results for FMI prototype, ITEA2 Modelisar internal paper, 2011
- [4] T. Blochwitz, T. Neidhold (ITI GmbH), M. Otter (DLR), M. Arnold (University of Halle), C. Bausch, M. Monteiro (Atego Systems GmbH), C. Claus, S. Wolf (Fraunhofer IIS, EAS), H. Elmqvist, H. Olsson (Dassault Systemes), A. Junghanns, J. Mauss (QTronic GmbH), D. Neumerkel (Daimler AG), J. Peetz (Fraunhofer SCAI): The Functional Mock-Up Interface, Modelica Conference 2011, Dresden
- [5] FMI-Homepage including FMI specifications www.fmi-standard.org

Using Functional Mock-up Units for Nonlinear Model Predictive Control

Manuel Gräber¹ Christian Kirches² Dirk Scharff³ Wilhelm Tegethoff^{1,3}

¹Technische Universität Braunschweig, Braunschweig, Germany

²Interdisciplinary Center for Scientific Computing (IWR), Heidelberg University, Germany

³TLK-Thermo GmbH, Braunschweig, Germany

Abstract

A software framework for prototyping of Nonlinear Model Predictive Control (NMPC) loops is presented that is based on the standardized model exchange format FMI (Functional Mock-up Interface). Arising optimal control problems are solved by an efficient implementation of the direct multiple shooting method, which is especially suitable for nonlinear and stiff system models. Using co-simulation, an optimizer, plant, and estimator can be coupled to a closed NMPC loop. Several stages of a typical control design process are supported, ranging from virtual simulation experiments to real plants with prototype NMPC controllers. Energy efficient control of vapor compression cycles is presented as example application of the proposed methods.

Keywords: Functional Mock-up Interface; Nonlinear Model Predictive Control; Vapor Compression Cycles

1 Introduction

Nonlinear Model Predictive Control (NMPC) provides promising possibilities to improve control accuracy, stability, as well as energy and economical efficiency of technical systems. The key idea is to utilize rigorous mathematical models of the controlled plant for online computations of appropriate control actions, based on the repeated solution of a dynamic optimization problem. Model-plant mismatch and disturbances are incorporated by updating the mathematical model according to estimates obtained from most recent measurement data. From the point of view of the numerical algorithms employed, these methods are well developed and ready to use. Their application to complex systems however by now is the subject of a few

selected research projects only. The most prevalent reason for this may well be the considerably large effort required to develop fast implementations of large-scale accurate nonlinear models. The development of object-oriented and equation-based modeling languages such as Modelica aims at helping to considerably reduce this effort: systems can be conveniently modeled by composition from smaller, reusable sub-components. Moreover, there no longer is the need to manually transform equations into a signal-oriented form.

In the last few years, Modelica has matured to a modeling language that is widely used for systems simulation in both academics and industry. More recently, research initiatives have come up that helped to extend the scope of Modelica beyond pure systems simulation. For example, [7] gives an overview over current research activities in the area, and shows possible further directions especially from a control design perspective.

Probably the first work reported in literature about dynamic optimization with Modelica models can be found in [12]. Therein the MATLAB S-Function format is used to interface Modelica models with an optimization solver. Dynamic optimization with models generated by the C-code export functionality of the Modelica tool Dymola is described in [16] and [25]. Both approaches suffer from the fact that the used model exchange formats are proprietary. In [26] the development of optimization based controllers in Modelica is addressed. But the authors remain unclear about the technical details how a Modelica model can be reused as internal model of the control algorithms.

A more integrated approach is described in [1]. Based on Optimica, a language extension of Modelica that serves to formulate optimization problems, an open source Modelica simulation and optimization tool has been developed that goes by the name *JMod-*

elica.org, see [2]. Therein, dynamic system models are formulated in the Modelica language and are symbolically transformed into a representation suitable for evaluation by numerical solvers. As is the case for most Modelica tools, not all parts of the Modelica language and the Modelica Standard Library are supported yet. Optimal control problems can be solved in *JModelica.org* by means of a *direct collocation* method.

As part of the ITEA-2 research project Modelisar, the standardized model exchange format FMI (*Functional Mock-up Interface*) [22, 3], has been developed. During the last two years, FMI gained a lot of attention and is now supported by over 20 simulation tools. A detailed list can be found on <http://fmi-standard.org>. The main purpose of FMI is to exchange models between different *simulation* tools. FMI is used to design nonlinear *Kalman Filters* for state and parameter estimation in [6]. To the best of our knowledge, there are no reports of FMI having been applied to *optimization* of dynamic systems, though.

1.1 Contribution

This article addresses the above described situation by presenting a software framework for fast and reliable prototyping of NMPC loops using the FMI standard [22]. The key idea is to use existing specialized software for each task and exchanging models between these tools, relying on FMI for the purpose. Using established modeling and simulation tools such as Dymola, one can conveniently set up large-scale and complex system models. Exported as FMI models, called FMUs (*Functional Mock-up Units*), we import these into the direct optimal control code MUSCOD-II [4, 9, 21]. MUSCOD-II is a software package for efficient numerical solution of optimal control problems. The implemented *direct multiple shooting method* is favorable especially for large-scale, highly non-linear, and stiff systems.

Using the co-simulation platform TISC [20], we also present a software solution for coupling optimization algorithms with simulation tools to conveniently test designed NMPC loops. Using existing interfaces to measurement and automation software NMPC controllers can also be connected to real plants.

With NMPC of a vapor compression cycle, we present a challenging but promising application and demonstrate the capability of our method.

1.2 Structure of the Paper

The paper starts with a description of the theoretical background of our methods. In Section 2 the underlying model class is defined. Based on this dynamic system model, a class of continuous *Optimal Control Problems* (OCPs) is formulated. The direct multiple shooting method is presented in Section 3 as an efficient numerical approach for the discretization and solution of OCPs. The control loop is closed in Section 4 by taking into account state estimates or measurements and repeatedly solving the OCP. In order to derive an efficient control algorithm, special attention is paid to reinitialization of subsequent optimization iterations and the separation of each iteration into different phases. Technical details of our methods are presented in Sections 5 and 6. We discuss optimization results for an example application in Section 7, using the presented toolchain and algorithms.

2 Problem Class

Starting point is an index-1 system of semi-explicit differential algebraic equations (DAE) describing the dynamic behavior of a controlled plant:

$$\frac{dx}{dt}(t) = f(x(t), z(t), u(t), p), \quad t \in \mathcal{T}, \quad (1a)$$

$$0 = g(x(t), z(t), u(t), p) \quad (1b)$$

with independent variable time t on the horizon $\mathcal{T} := [0, t_f]$, differential state variables $x(\cdot) \in \mathbb{R}^{n_x}$, algebraic state variables $z(\cdot) \in \mathbb{R}^{n_z}$, control functions $u(\cdot) \in \mathbb{R}^{n_u}$ and time-invariant model parameters $p \in \mathbb{R}^{n_p}$. Later on, we will show how to use the FMI [22] to conveniently exchange models of type (1) between different modeling software tools.

We may then formulate an OCP based on plant model (1) to find locally optimal control trajectories on the time horizon \mathcal{T} for a given initial process state x_0 . To this end, we need to express the performance measure as an OCP objective function, i.e., a combination of a Lagrange-type term L ,

$$\int_0^{t_f} L(x(t), z(t), u(t), p) dt, \quad (2)$$

and a Mayer-type term E that is defined at the end of time horizon only,

$$E(x(t_f), z(t_f), p). \quad (3)$$

With the resulting objective function

$$\Phi(x(\cdot), u(\cdot), z(\cdot), p) := \int_0^{t_f} L(x(t), z(t), u(t), p) dt \quad (4)$$

$$+ E(x(t_f), z(t_f), p),$$

an OCP can be formulated as follows:

$$\min_{\substack{x(\cdot), z(\cdot), \\ u(\cdot), p}} \Phi(x(\cdot), z(\cdot), u(\cdot), p) \quad (5a)$$

$$\text{s.t. } \frac{dx}{dt}(t) = f(x(t), z(t), u(t), p), t \in \mathcal{T}, \quad (5b)$$

$$0 = g(x(t), z(t), u(t), p), t \in \mathcal{T}, \quad (5c)$$

$$0 \leq c(x(t), z(t), u(t), p), t \in \mathcal{T}, \quad (5d)$$

$$0 \leq r_i(x(t_i), z(t_i), p), \quad \{t_i\}_i \subset \mathcal{T}, \quad (5e)$$

$$0 = x(0) - x_0. \quad (5f)$$

We strive to identify trajectories for the controls $u(\cdot)$ and the differential and algebraic states $(x(\cdot), z(\cdot))$ that minimize the cost function Φ , and are a solution to the initial value problem defined by (5b) and (5f). Additionally, mixed state-control inequality constraints (5d) and point constraints on a grid $\{t_i\}_i \subset \mathcal{T}$ (5e) must be satisfied.

3 Direct Multiple Shooting

The OCP presented in Section 2 is an infinite-dimensional optimization problem. The purpose of the Direct Multiple Shooting method [4, 21] is to transform this problem into a finite dimensional nonlinear program (NLP) by discretization of the control functions and path constraints and by parameterization of the state trajectories. To this end, we introduce a *shooting grid* $\{\tau_i\}_{0 \leq i \leq N}$,

$$0 = \tau_0 < \tau_1 < \dots < \tau_N = t_f. \quad (6)$$

on the horizon \mathcal{T} . Control trajectories are discretized on the shooting grid, e.g. as piecewise constant functions

$$u(t) := u_i, \quad t \in [\tau_i, \tau_{i+1}) \subset \mathcal{T}, \quad 0 \leq i \leq N-1. \quad (7)$$

The control space is hence reduced to functions depending on finitely many parameters u_i only.

Multiple shooting state variables s_i are introduced on the time grid to parameterize the differential state trajectories. The node values serve as initial values for an IVP solver computing the state trajectories independently on the shooting intervals $0 \leq i < N$,

$$\frac{dx_i}{dt}(t) = f(x_i(t), z_i(t), u_i, p), \quad t \in [\tau_i, \tau_{i+1}] \quad (8a)$$

$$0 = g(x_i(t), z_i(t), u_i, p), \quad (8b)$$

$$x_i(\tau_i) = s_i. \quad (8c)$$

Obviously we obtain from the above IVPs N trajectories, which in general will not combine to a single continuous trajectory. Continuity across shooting intervals needs to be ensured by additional matching conditions entering the NLP as equality constraints,

$$s_{i+1} = x_i(\tau_{i+1}; \tau_i, s_i, z_i, u_i, p), \quad 0 \leq i \leq N-1. \quad (9)$$

Here we denote by $x_i(\tau_{i+1}; t_i, s_i, z_i, u_i, p)$ the solution of the IVP on shooting interval i , evaluated in τ_{i+1} , and depending on the initial time t_i , initial states (s_i, z_i) , and on control and model parameters u_i and p . Path constraints $c(\cdot)$ are discretized on the shooting grid for simplicity of exposition. Likewise, the point constraint grid is assumed to coincide with the shooting grid.

From this discretization and parameterization, we obtain a highly structured NLP of the form

$$\min_{\xi} \sum_{i=0}^N l_i(\tau_i, s_i, z_i, u_i, p) \quad (10a)$$

$$\text{s.t. } s_{i+1} = x_i(\tau_{i+1}; \tau_i, s_i, z_i, u_i, p) \quad 0 \leq i < N, \quad (10b)$$

$$0 = g(\tau_i, s_i, z_i, u_i, p), \quad 0 \leq i \leq N, \quad (10c)$$

$$0 \leq c(\tau_i, s_i, z_i, u_i, p) \quad 0 \leq i \leq N, \quad (10d)$$

$$0 \leq r_i(\tau_i, s_i, z_i, u_i, p) \quad 0 \leq i \leq N, \quad (10e)$$

$$0 = s_0 - x_0, \quad (10f)$$

where all unknowns of the problem are grouped in a single vector $\xi := (s_0, z_0, \dots, s_N, z_N, u_0, \dots, u_{N-1})$. For the ease of notation, we write $u_N := u_{N-1}$ in (10).

We solve this large-scale but structured NLP by a tailored sequential quadratic programming (SQP) method. This includes an extensive exploitation of the arising structures, in particular using block-wise high-rank updates of the Hessian approximation, a partial null-space reduction to eliminate the algebraic states [21], and condensing techniques for a reduction of the size of this QP to the dimension of the initial values s_0 and controls (u_0, \dots, u_{N-1}) only [4, 21].

Note that the evaluation of the matching condition constraint (10b) requires the solution of an initial value problem with initial values (s_i, z_i) and controls u_i on the time horizon $[\tau_i, \tau_{i+1}]$. For more details on the numerical algorithms and techniques employed we refer the reader to e.g. the textbook [24] for nonlinear programming in general, and to [4, 21] for details on nonlinear programming techniques for Direct Multiple Shooting. An efficient implementation is available with the software package MUSCOD-II [9, 21] that has been used for all computations. MUSCOD-II for off-line optimal control is publicly available [19] on the *NEOS Server for Optimization* [15].

4 Nonlinear Model Predictive Control Scheme

We now address the issue of solving OCP (10) in an on-line NMPC setting. Key to an efficient numerical algorithm for NMPC is to reuse in every iteration information available from the last problem's solution to initialize the new problem. This is due to the fact that subsequent problems differ only in the real-world process state x_0 (5f). Moreover, the faster the control feedback can be computed and applied to the real-world process, the more similar the subsequent problems will be. If model predictions are sufficiently close to real process behavior, it is reasonable to expect that the information contained in the previous problem's solution already is a very good initial guess close to the desired solution of the new subproblem.

4.1 Initial Value Embedding

In [8, 9] and subsequent works it has been proposed to initialize the current problem with the full solution of the previous optimization run, in particular control variables u_i and state variables (s_i, z_i) . We refer to [10] for a detailed survey on the topic of initial value embedding. It is a prominent feature of the Direct Multiple Shooting approach that very good state initializers are available not only for $x(0)$ but also for the shooting grid nodes $x(\tau_i)$, $1 \leq i \leq N$.

In using the proposed initialization, the value of s_0 will in general *not* be the value of the current state x_0 . By explicitly including the linear initial value constraint (10f) we can however guarantee that s_0 attains the value of x_0 already after the first full Newton-type step computed by the SQP method.

4.2 A Real-Time Iteration in Three Phases

This idea motivates the idea of *real-time iterations* that perform only one SQP iteration per NMPC sample [9]. In this iteration, we can evaluate all derivatives and all function values without requiring knowledge of the current state x_0 , the only exception being the linear initial value constraint. Consequently, we can pre-solve a major part of the direct multiple shooting SQP step as follows:

Preparation All functions and derivatives that do not require knowledge of x_0 are evaluated. This includes ODE solution, sensitivity computation, sparsity analysis, structure exploitation, and matrix factorizations. Note that the preparation

phase of the new problem always takes place one sampling period ahead.

Feedback As soon as x_0 is available, the SQP step is readily computed from the prepared data, but only as far as required to give a feedback control to the process. Hence, the feedback delay reduces to the computation time of the SQP step *after preparation* that essentially involves the solution of only a single QP.

Transition The SQP step computation is completed *after* the feedback control has been given to the process.

5 Software Framework

In this section we present our software framework for a convenient setup of simulated and real-world NMPC loops. The basic idea is to use different specialized software for each task and to couple it to a co-simulation master. Using FMI ensures integrity of the underlying plant model that is used in several places, and avoids error-prone and time-consuming model transformations.

5.1 General Structure

A closed NMPC loop consists of three major parts as sketched in Figure 1:

Plant The controlled system. This could be a real-world plant or, in an earlier design stage, a virtual plant based on a simulation model.

Estimator The current value of all state values and parameters of the system model is estimated from available measurement data $y(t)$. The estimator could be realized as a nonlinear Kalman filter or a moving horizon estimator (MHE). If a virtual plant is used wherein all state variables and parameters are accessible, it is also possible to use an ideal estimator with $(x(t), p) = y(t)$.

Optimizer The heart of an NMPC loop is an optimization algorithm that determines the best possible control action for the current system state. This is realized as described in Sections 3 and 4.

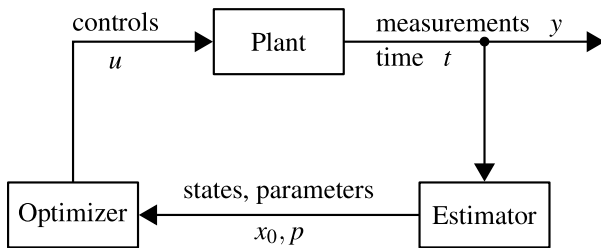


Figure 1: Signal flow diagram of closed NMPC loop.

5.2 Data Exchange

We use the co-simulation platform TISC [20] to set-up a powerful NMPC prototyping environment, keeping the basic structure of Figure 1 in mind. TISC acts as master and manages data exchange between different clients. There already exist interfaces between TISC and a variety of simulation, visualization and measurement tools, e.g. Dymola, LabView, and Simulink. The user has to define types and names of variables to be sent and received for each client. Data routing between clients is automatically managed by matching variable types and names. For our NMPC environment we use a fixed naming and typing convention. Variable names and the direction of information flow are defined according to Figure 1. The TISC type of *time* is *Double*, whereas all other variables are of TISC type *DoubleArray*.

Using this definition it is readily possible to exchange components of an NMPC loop. For example, one could replace a virtual plant that is simulated in Dymola with a real plant interfaced through LabView with just a few mouse clicks.

6 FMI for Optimization

In this section we show some implementation details to shed light on how an FMU can be used in MUSCOD-II to formulate and solve OCPs of type (10). We also describe new requirements and demands the FMI standard faces when we desire to use in a consistent derivative-based optimization setting such as direct optimal control, and give recommendations on future enhancements of FMI.

6.1 Interface between MUSCOD and FMI

In order to set up and solve a OPC in MUSCOD-II the user has to provide a C++ file that defines the model equations, including differential equations, objectives, and constraints. This source code is compiled into

a shared library and dynamically loaded by the main program MUSCOD-II during runtime.

Instead of modeling in C++, we link a compiled FMU to a generic MUSCOD-II model that calls the appropriate FMI functions. This paradigm has also been followed by [19] to interface MUSCOD-II with AMPL [11]. As defined in FMI, some function calls have to be carried out once during startup in order to instantiate and initialize an FMU. This is organized by defining a class, writing the required function calls in its constructor, and instantiating it as a global object. Now, the constructor is called when the resulting dynamic library is loaded into MUSCOD-II. The corresponding code is shown in Listing 1. The pointer to the instantiated FMU is defined globally, because we need to call FMU functions in several places.

```

#define NXD 19
#define NU 2
#define NP 0

fmiComponent fmu;
const fmiValueReference uRef[NU] =
    {352321536, 352321537};

class InstantiateFMU {
public:
    InstantiateFMU();
    ~InstantiateFMU();
};

InstantiateFMU::InstantiateFMU ()
{
    // Instantiate fmu
    fmu = fmiInstantiateModel (instanceName,
        GUID, callbacks, fmiFalse);
    // Set Time
    status = fmiSetTime(fmu, 0.0);
    // Set Controls
    const fmiReal uIni[NU] = {2.5, 41.6667};
    status = fmiSetReal (fmu, uRef, NU, uIni);
    // Set Parameters
    const fmiReal pInit[NP] = {};
    fmiSetReal(fmu, pRef, NP, p);
    // Initialize
    fmiEventInfo eventInfo;
    status = fmiInitialize(fmu, fmiFalse, 0.0,
        &eventInfo);
}

InstantiateFMU instantiateFMU;
  
```

Listing 1: Instantiation and initialization of a FMU in a MUSCOD model source file.

First of all we have to provide the differential right-hand side function of the ODE, as shown in Listing 2. This function is called by a MUSCOD-II integrator and is expected to return the right-hand as a function of time, states, controls, and parameters. The objective function is formulated in a similar way. As an example, the source code of a Lagrange term is shown in Listing 3.

```

void ffcn (
  double *t,    double *xd,    double *xa,
  double *u,    double *p,    double *rhs,
  double *rwh, long  *iwh,    long  *info
) {
  // Set Time
  fmiSetTime (fmu, *t);
  // Set Controls
  fmiSetReal (fmu, uRef, NU, u);
  // Set Parameters
  fmiSetReal (fmu, pRef, NP, p);
  // Set States
  fmiSetContinuousStates (fmu, xd, NXD);
  // Get Derivatives
  fmiGetDerivatives (fmu, rhs, NXD);
}

```

Listing 2: Right-hand side function.

```

void lfcn (
  double *t,    double *xd,    double *xa,
  double *u,    double *p,    double *lval,
  double *rwh, long  *iwh,    long  *info
) {
  // Set Time
  fmiSetTime (fmu, *t);
  // Set Inputs
  fmiSetReal (fmu, uRef, NU, u);
  // Set States
  fmiSetContinuousStates (fmu, xd, NXD);
  // Get Outputs
  const fmiValueReference yRef[2] =
    {905970080, 905971331};
  double y[2];
  fmiGetReal (fmu, yRef, 2, y);

  *lval = (y[1]-283.15) * (y[1]-283.15)
    + 0.01 * y[0] / 1000.0;
}

```

Listing 3: Lagrange term of objective.

A large part of this source code can be generated automatically from the model description xml file of an FMU, but some lines, e.g. objective formulation, currently still need to be coded by hand.

6.2 Directions for Future FMI Developments

In this section we give an outlook on future developments in using FMI for direct dynamic optimization. Ideally, we are interested in realizing FMI access to the full class of DAE-constrained switched systems,

$$\frac{dx}{dt}(t) = f_{\sigma}(t, x(t), z(t), u(t), p), \quad t \in \mathcal{T}, \quad (11a)$$

$$0 = g_{\sigma}(t, x(t), z(t), u(t), p), \quad (11b)$$

$$\sigma_i(t) = \begin{cases} +1 & s(t, x(t), z(t), u(t), p) > 0, \\ -1 & s(t, x(t), z(t), u(t), p) < 0. \end{cases}, \quad (11c)$$

$$i = 1, \dots, n_{\sigma}.$$

Additional transversality conditions must be satisfied to guarantee that points $s(t, x(t), z(t), u(t), p) = 0$ are

isolated and a clear transition between the two alternate right-hand sides occurs in the neighborhood of such points, see e.g. [5].

The principle of internal numerical differentiation (IND) requires a caller-control approach to be used for consistent derivative-based optimization. In such an approach, FMI is responsible for evaluation of the functions f and g , if given a *caller-supplied* switch signature σ , factorization of $\frac{dg}{dz}$, iteration count for solving the DAE constraint $0 = g(\cdot)$, etc. The caller is then able to keep these potentially nondifferentiable parts of the evaluation of system (11) fixed for the purpose of computing consistent derivatives and sensitivities of IVP solutions, e.g., as described in [5, 18, 23] for the case of implicit switches.

6.3 FMI Requirements for Consistent Derivatives

The current implementation of the FMI standard has proven sufficient to enable our tools to work with FMI when the problem class is limited to continuous ODEs. DAEs are currently handled internally, and are exposed as ODEs in a reduced space to the caller. This involves iterative solution of the nonlinear DAE constraint that is carried out internally by the FMI. Implicitly discontinuous ODEs, so-called switched or hybrid systems, are supported in an accessible way by the FMI standard. State discontinuities however are handled internally again. This effectively limits our approach to FMI for optimization to ODEs with continuous solutions.

To extend the FMI standard to complement state-of-the-art optimization software, the paradigm of external control over adaptive components needs to be adhered to. This currently is partially the case for switched systems, but needs to be extended to, e.g., state discontinuities, direct linear algebra involving pivoting decisions, and to the use of iterative solvers.

Whenever it is desirable to call such procedures inside an FMI model, all information about control about adaptive components, including pivoting sequences, iteration counts, matrix factors, outcome of conditional evaluations, or choice between alternate functions during implicit switching, should be conveyed to the FMI by the caller. This would grant the caller control over potential sources of non-differentiability inside the FMI. We propose that the caller should maintain an *FMI state object* that documents the state and outcome of all non-differentiable actions, and would pass this FMI state object to the FMI, to be used for subsequent function evaluation. The caller would fur-

ther modify this FMI state object whenever appropriate, e.g. exchange functions during implicit switching, but only after the arising non-differentiability or discontinuity has been taken care of on the optimizer's side. Indeed, the FMI 2.0 standard makes considerable progress into this direction.

7 Example Application: Vapor Compression Cycle

To illustrate the applicability of the NMPC tools and algorithms described in the previous sections, we present simulation results for a challenging NMPC application. We desire to control a vapor compression cycle with two goals in mind: good disturbance rejection and maximum energy efficiency.

7.1 System Description

The system under consideration is sketched in Figure 2. Main components are two plate heat exchangers, a variable-speed scroll compressor, an electronic expansion valve and a suction line accumulator. Working fluids are internally refrigerant R134a and on both secondary sides water-glycol mixtures. This system also exists in reality and is designed as test stand for automotive air-conditioning compressors.

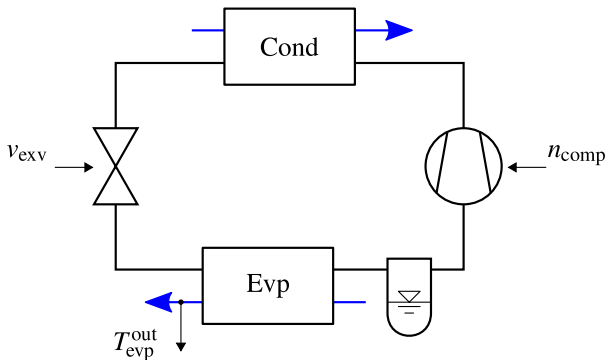


Figure 2: Vapor compression cycle including inputs and controlled outputs of the system.

7.2 System Model and Optimal Control Problem

The system model is derived from first principles only. The condenser is modeled as moving boundary model, details can be found in [14]. Accumulator and evaporator are modeled as lumped volumes.

Refrigerant fluid properties are incorporated using bicubic spline interpolation. This approach leads to

improved computational speed and smoothness compared to the commonly used iterative solution of fundamental equations. Further information can be found in [13].

The resulting system model is an explicit ODE system with 17 differential states. There are 2 controls: a voltage signal v_{exv} to the step motor controller actuating the expansion valve and the rotational speed set-point of the compressor n_{comp} .

The main control goal is to keep the evaporator outlet water temperature $T_{\text{evp}}^{\text{out}}$ at a fixed set point T_{set} . We formulate the squared deviation as first Lagrange-type objective term:

$$\int_0^{t_f} (T_{\text{evp}}^{\text{out}}(t) - T_{\text{set}})^2 dt. \quad (12)$$

We also want to maximize energy efficiency, in other words, minimize the electrical power P_{comp} needed by the compressor, leading to the second Lagrange-type objective term:

$$\int_0^{t_f} P_{\text{comp}}(t) dt. \quad (13)$$

We also desire to realize a smooth control profile by penalizing excessive control action, adding

$$\int_0^{t_f} (n_{\text{comp}} - \tilde{n})^2 dt, \quad (14)$$

$$\int_0^{t_f} (v_{\text{exv}} - \tilde{v})^2 dt \quad (15)$$

to our objective. Where \tilde{n} and \tilde{v} are two additional state variables the original ODE system is augmented by. The corresponding additional equations are

$$\frac{d\tilde{n}}{dt} = n_{\text{comp}} - \tilde{n}, \quad (16)$$

$$\frac{d\tilde{v}}{dt} = v_{\text{exv}} - \tilde{v}. \quad (17)$$

Weighting factors w_i are introduced and all terms are combined to the objective

$$\begin{aligned} \Phi := \int_0^{t_f} & [(T_{\text{evp}}^{\text{out}}(t) - T_{\text{set}})^2 + w_0 P_{\text{comp}}(t) \\ & + w_1 (n_{\text{comp}} - \tilde{n})^2 + w_2 (v_{\text{exv}} - \tilde{v})^2] dt \end{aligned} \quad (18)$$

We finally obtain a OCP of type

$$\min_{x(\cdot), u(\cdot)} \Phi(x(\cdot), u(\cdot)) \quad (19a)$$

$$\text{s.t. } \frac{dx}{dt}(t) = f(x(t), u(t)) \quad t \in \mathcal{T}, \quad (19b)$$

$$0 \leq c(x(t), u(t)) \quad t \in \mathcal{T}, \quad (19c)$$

$$0 = x(0) - x_0, \quad (19d)$$

with 19 differential states x and 2 controls u . In addition to the plant model ODE (19b), fixed upper and lower bounds for all states and controls (19c) as well as initial values for all states (19d) are considered.

7.3 Simulation Results – NMPC

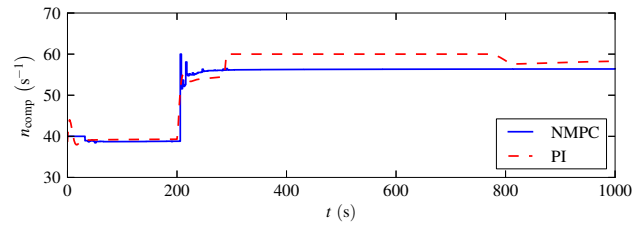
Using the methods and software tools described in previous sections we can set up a closed loop NMPC simulation. The vapor compression system Modelica model is developed, and exported as an FMU using Dymola. As described in section 6.1, the exported FMU is used in MUSCOD-II to formulate and solve the arising optimal control problems of type (19).

Investigation of a range of choices for the NMPC controller's parameters, comprising time horizon, number of multiple shooting intervals, and sampling rate, leads to the final choice of 500 s time horizon divided into 10 multiple shooting intervals and a 2 s sampling interval of the closed loop controller. Control trajectories are discretized on the same grid by piecewise constant functions. This setup shows good closed loop performance in terms of stability and disturbance rejection.

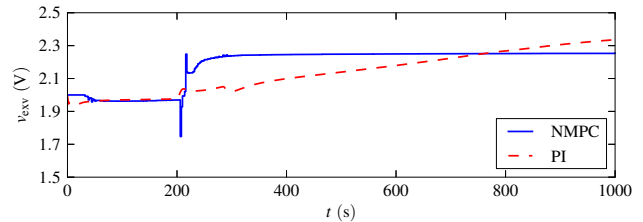
The chosen prediction horizon of 500 s appears to be very large at first sight, but shorter prediction horizons have been found to lead to stability issues. This behavior is mathematically explained by large time constants of the system. A physical explanation can be given by a closer look at the suction line accumulator. In this component, liquid refrigerant is stored in order to compensate for changes of the optimal active refrigerant charge at different working points; see [17] for a detailed discussion. The second task of a suction line accumulator is to separate vapor from liquid and feed the compressor with pure vapor. In steady-state conditions for the whole cycle, the accumulator energy balance forces inlet and outlet refrigerant states to an equilibrium. The accumulator can therefore be seen to act as a passive control unit that drives two points of the cycle (accumulator inlet and outlet) to the dew line. This passive control action takes place comparatively slowly, resulting in large time constants of the system model.

A virtual NMPC experiment is set up by simulating the controlled plant in Dymola and coupling it with MUSCOD-II via TISC. The real-time iteration scheme presented in Section 4.2 is applied with a fixed sampling rate of 2 s, assuming zero feedback delay.

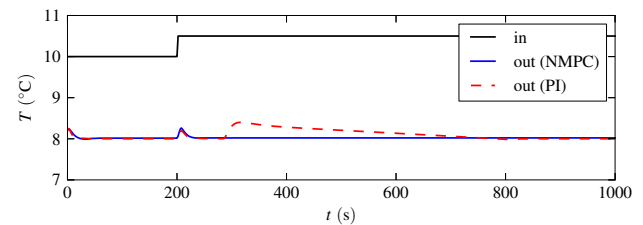
Additional assumptions are no model-plant mismatch, availability of the full process state vector, and uncontrolled input measurements without disturbance.



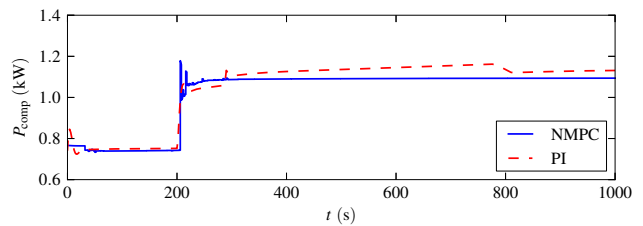
(a) Control input 1: compressor speed.



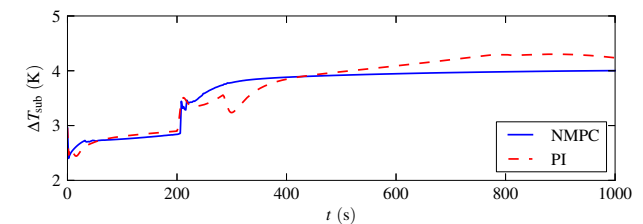
(b) Control input 2: expansion valve voltage signal.



(c) Chilled water temperatures at evaporator inlet and outlet.



(d) Compressor's electrical power consumption.



(e) Refrigerant subcooling at condenser outlet.

Figure 3: Simulation results: NMPC versus PI control of a vapor compression cycle.

Although these assumptions can hardly be satisfied when NMPC is applied to a real plant, this kind of ideal experiment still helps to gain insight into the theoretical performance of an optimally designed NMPC controller. Using our software framework, closed loop performance of extended problems can be studied very conveniently.

7.4 Simulation Results – Comparison to PI Control

For comparison, we applied a conventional control concept with two continuous PI controllers to the plant. Our primary goal – keeping chilled water outlet temperature at a constant set-point of 8 °C – is achieved by adjusting compressor speed. Contrary to NMPC, we can't take our second goal – maximizing energy efficiency – directly into account. It is known, however, that for vapor compression cycles of our type, a certain value of refrigerant subcooling at the condenser outlet is optimal [17]. Hence, we may use the second control input – expansion valve opening – to keep subcooling close to a fixed set-point of 3 K.

In our example experiment we start with near steady-state conditions. At $t = 200$ s the chilled water inlet temperature rises from 10 to 10.5 °C. With chilled water outlet at 8 °C, this results in a cooling load increase of 25%. Figure 3 shows the corresponding response of PI and NMPC closed loops.

In the first 200 seconds there is only little control action. Both control goals, chilled water outlet temperature (Figure 3(c)) and compressor's power consumption (Figure 3(d)), are almost identical for both control concepts. This is because the chosen subcooling setpoint for the PI controller is set to 3 K, which is close to the efficiency optimal working point for these boundary conditions. At $t = 200$ s, when the chilled water inlet temperature rises, things change noticeably. First of all, there is an immediate deviation of the chilled water outlet temperature from its setpoint. Both controllers react by increasing the compressor speed (Figure 3(a)) and drive the temperature back to their setpoints (Figure 3(c)). Looking at Figure 3(b), we see that both controllers react to the disturbance by opening the expansion valve. The NMPC controller however does so much more aggressively, leading to the desired result that water outlet temperature stays at its setpoint for the remaining simulation time. The PI controlled temperature shows a second deviation starting at $t = 300$ s. Because the maximum compressor speed of 60 s⁻¹ has already been reached, the temperature deviation lasts until $t = 800$ s.

One could argue that the situation could be improved by tuning the expansion valve PI controller to speed up its reaction. Although we don't claim to have chosen the best possible PI parameters, simulation studies show that the expansion valve PI controller must be comparatively slow in order to ensure stability of the closed loop. This may be due to the large time constants mentioned above. A second reason may

be the inverse response behavior of the plant model for expansion valve opening as input and subcooling as output. Besides good disturbance rejection, a second benefit of our NMPC controller becomes clear by looking at the compressor power consumption in Figure 3(d). At $t = 1000$ s the system slowly approaches a new steady state working point with about 4% increased power consumption of the PI controlled compared to the NMPC controlled cycle. Therefore, one can see that a fixed subcooling setpoint is not optimal for all boundary conditions. Figure 3(e) shows that for the new working point, optimal subcooling tracked by the NMPC controller lies somewhere around 4 K. If we continued simulation, the PI controller would steer the cycle back to non-optimal subcooling of 3 K.

8 Conclusion

Although tailored to forward simulation, the FMI format can be used for interfacing Modelica models with state-of-the-art dynamic optimization software. But with the current design of FMI this approach is limited to continuous ODE. To extend the scope of FMI for optimization to hybrid DAE there must be major changes. Instead of solving implicit algebraic equations with embedded solvers internally, the residuum functions should be exposed. The proposed software framework has proven its applicability for setting up NMPC loops in an early design stage. The application vapor compression cycle demonstrates the benefits of NMPC. In the presented scenario, NMPC shows a significantly better performance compared to a conventional PI control concept in terms of energy efficiency and disturbance rejection. Moreover, NMPC is able to identify and track new optimal working points under changed external conditions.

References

- [1] Johan Åkesson. *Languages and Tools for Optimization of Large-Scale Systems*. Phd thesis, Lund University, 2007.
- [2] 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 & Chemical Engineering*, 34(11):1737–1749, November 2010.
- [3] T. Blochwitz, M. Otter, M. Arnold, C. Bausch, C. Clauß, H. Elmqvist, A. Junghanns, J. Mauss, M. Monteiro, T. Neidhold, D. Neumerkel, H. Olsson, J.-V. Peetz, and S. Wolf. The Functional Mockup Interface for Tool independent Exchange of Simulation Models. In *8th International Modelica Conference*, Dresden, 2011.

- [4] H. G. Bock and K. J. Plitt. A Multiple Shooting algorithm for direct solution of optimal control problems. In *Proceedings 9th IFAC World Congress Budapest*, pages 243–247. Pergamon Press, 1984.
- [5] H.G. Bock. Numerical treatment of inverse problems in chemical reaction kinetics. In K.H. Ebert, P. Deuffhard, and W. Jäger, editors, *Modelling of Chemical Reaction Systems*, volume 18 of *Springer Series in Chemical Physics*, pages 102–125. Springer, Heidelberg, 1981.
- [6] Jonathan Brembeck, Martin Otter, and Dirk Zimmer. Nonlinear Observers based on the Functional Mockup Interface with Applications to Electric Vehicles. In *8th International Modelica Conference*, Dresden, 2011.
- [7] Francesco Casella, Filippo Donida, and Marco Lovera. Beyond Simulation: Computer-Aided Control System Design using Equation-based Object-oriented Modelling for the Next Decade. *Simulation News Europe*, 19(1):29–41, 2009.
- [8] M. Diehl, H. G. Bock, J. P. Schlöder, R. Findeisen, Z. Nagy, and F. Allgöwer. Real-time optimization and Nonlinear Model Predictive Control of Processes governed by differential-algebraic equations. *Journal of Process Control*, 12(4):577–585, 2002.
- [9] Moritz Diehl. *Real-Time Optimization for Large Scale Nonlinear Processes*. Phd thesis, Universität Heidelberg, 2001.
- [10] Moritz Diehl, Hans Joachim Ferreau, and Niels Haverbeke. Efficient Numerical Methods for Nonlinear MPC and Moving Horizon Estimation. In Lalo Magni, Davide Martino Raimondo, and Frank Allgöwer, editors, *Nonlinear Model Predictive Control*, Lecture Notes in Control and Information Sciences, pages 391–417. Springer, Berlin, Heidelberg, New York, 2009.
- [11] R. Fourer, D. M. Gay, and B. W. Kernighan. *AMPL: A Modelling Language for Mathematical Programming*. Books/Cole—Thomson Learning, 2nd edition, 2003.
- [12] Rüdiger Franke. Formulation of dynamic optimization problems using Modelica and their efficient solution. In *2nd International Modelica Conference*, pages 315–323, Oberpfaffenhofen, 2002.
- [13] Manuel Gräber, Christian Kirches, Johannes P. Schlöder, and Wilhelm Tegethoff. Nonlinear Model Predictive Control of a Vapor Compression Cycle based on First Principle Models. In *MATHMOD, 7th Vienna International Conference on Mathematical Modelling*, 2012.
- [14] Manuel Gräber, Nils Christian Strupp, and Wilhelm Tegethoff. Moving boundary heat exchanger model and validation procedure. In *EUROSIM Congress on Modelling and Simulation*, Prague, 2010.
- [15] William Gropp and Jorge J. Moré. Optimization environments and the NEOS Server. In M. D. Buhmann and A. Iserles, editors, *Approximation Theory and Optimization*, pages 167–182. Cambridge University Press, 1997.
- [16] L. Imsland, P. Kittilsen, and T. S. Schei. Model-based optimizing control and estimation using Modelica models. *Modeling, Identification and Control*, 31(3):107–121, 2010.
- [17] Jørgen Bauck Jensen and Sigurd Skogestad. Optimal operation of simple refrigeration cycles Part I: Degrees of freedom and optimality of sub-cooling. *Computers & Chemical Engineering*, 31(5-6):712–721, May 2007.
- [18] C. Kirches. A numerical method for nonlinear robust optimal control with implicit discontinuities and an application to powertrain oscillations. Diploma thesis, Ruprecht–Karls–Universität Heidelberg, October 2006.
- [19] Christian Kirches and Sven Leyffer. TACO – A Toolkit for AMPL Control Optimization. Preprint ANL/MCS-P1948-0911, Mathematics and Computer Science Division, Argonne National Laboratory, October 2011.
- [20] Roland Kossel, Martin Löffler, Nils Christian Strupp, and Wilhelm Tegethoff. Distributed energy system simulation of a vehicle. In *Vehicle Thermal Management Systems Conference*. Institution of Mechanical Engineers, SAE International, 2011.
- [21] D B Leineweber, I Bauer, A A S Schäfer, H G Bock, and J P Schlöder. An Efficient Multiple Shooting Based Reduced SQP Strategy for Large-Scale Dynamic Process Optimization (Parts I and II). *Computers and Chemical Engineering*, 27:157–174, 2003.
- [22] MODELISAR. *Function Mock-up Interface for Model Exchange*, 2010. Specification, Version 1.0.
- [23] K.D. Mombaur. *Stability Optimization of Open-loop Controlled Walking Robots*. Phd thesis, Universität Heidelberg, 2001.
- [24] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, Berlin, Heidelberg, New York, 2nd edition, 2006.
- [25] Andreas Pfeiffer. *Numerische Sensitivitätsanalyse unstetiger multidisziplinärer Modelle mit Anwendungen in der gradientenbasierten Optimierung*. Phd thesis, Martin-Luther-Universität Halle-Wittenberg, 2008.
- [26] E. D. Tate, Michael Sasena, Jesse Gohl, and Michael Tiller. Model Embedded Control: A Method to Rapidly Synthesize Controllers in a Modeling Environment. In *6th International Modelica Conference*, pages 493–502, Bielefeld, 2008.

Modeling a Low-temperature Compressed Air Energy Storage with Modelica

Marcus Budt¹, Daniel Wolf¹, Roland Span²

¹Fraunhofer Institute for Environmental, Safety, and Energy Technology UMSICHT
Osterfelder Straße 3, 46047 Oberhausen, Germany
marcus.budt@umsicht.fraunhofer.de, daniel.wolf@umsicht.fraunhofer.de

²Ruhr-University Bochum, Faculty for Mechanical Engineering
Universitätsstraße 150, 44780 Bochum, Germany
roland.span@thermo.rub.de

Abstract

The paper deals with the simulation of an innovative adiabatic compressed air energy storage plant. These plants are able to store electrical energy by compressing and expanding ambient air. In contrast to other approaches the plant layout examined in this paper works with much lower storage temperatures of just 100-200 °C. Aim of the modeling effort is to dynamically simulate the plant and to analyze the thermodynamics of the system. Here, off-design behavior regarding turbomachinery output temperatures, pressure losses and heat flows are of particular interest.

Keywords: compressed air; energy storage; thermal storage; low-temperature; CAES; modeling; Modelica

1 Introduction

The increasing share of renewable power generation, particularly of fluctuating wind and solar generation, leads to a time-based shift between supply and demand. A result of this development is the increasing demand for energy storage. Beside short time storage technologies like batteries or flywheels, a significant demand for bulk storage like pumped hydro energy storages (PHES) arises. For Europe the future PHES potential is rather limited due to siting restrictions including proper topological conditions. One alternative is compressed air energy storage (CAES), which provides energy capacities and power ranges comparable to those of PHES. This renders CAES a promising option for bulk electricity storage in the near term future.

2 Compressed Air Energy Storage

The idea of using compressed air to store energy is rather old. Beside pressurized air driven vehicles for special applications, there are two so called diabatic CAES plants, which are already in operation. The first CAES in Huntorf (Germany) works since 1978. The second one, located in McIntosh (USA), is in service since 1991. The concept of CAES is to absorb electricity by compressing ambient air by an electrically driven compressor in times of surplus electricity in the grid. The compressed air can be stored in a pressurized containment of any kind. The mentioned CAES plants use solution mined underground salt caverns as compressed air storage (CAS). Because of the surrounding salt these caverns are technically tight without additional sealing. During discharge the compressed air is released from the CAS and heated up to drive an expansion turbine. The expansion turbine is connected to a generator supplying electric power to the grid.

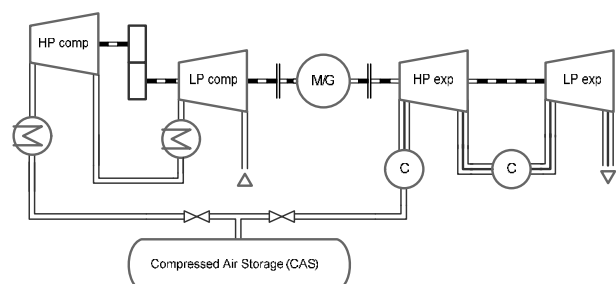


Figure 1: Block diagram of the first compressed air energy storage plant located in Huntorf, Germany [1]

As shown on the left hand side of Figure 1 the whole amount of heat generated during compression is cooled to the ambient in today's diabatic CAES. Therefore, two main intercoolers are installed in the Huntorf plant, the first one between the low and high

pressure compressor units and the second one between high pressure compressor unit and CAS. The second intercooler ensures that the air enters the CAS at a maximum temperature of 35 °C, because higher temperatures would destabilize it. In expansion mode these plants use a gas fired combustion chamber to pre-heat the compressed air before the expansion in order to protect the turbine and to increase the power output. In the Huntorf plant this pre-heating again is located at two points of the process. First, the air is pre-heated before entering the pressure expander and then again between the two expander units. Therefore, both diabatic CAES plants are, in the proper meaning of the word, no pure energy storages. They are rather a type of hybrid gas plants.

2.1 Current adiabatic design approaches

Nowadays CAES approaches aim on cycle operation without the need of fossil fuels to heat up the compressed air during expansion. Therefore, a thermal energy storage (TES) is applied. It captures the heat of compression during the charging process and allows for using it to heat up the air in the discharging process. Figure 2 shows the block diagram of an adiabatic compressed air energy storage (A-CAES).

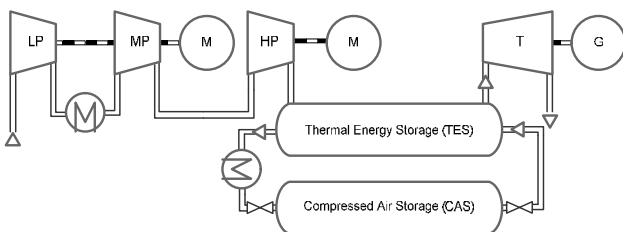


Figure 2: Concept of an adiabatic compressed air energy storage

Advantages of the concept are the high cycle efficiency of up to 70 % and the high energy density of the TES [2]. The main challenges are the demand for a compressor redesign to face temperatures of up to 650 °C and the development of a large packed bed TES, which can withstand high temperatures and pressures of around 70 bar simultaneously.

2.2 Low-temperature concept

To avoid the previously mentioned challenges Fraunhofer UMSICHT investigates the possibility to design A-CAES plants for lower TES temperatures. Interesting results for a two-stage A-CAES at 350 °C [1] and the fact that the cycle efficiency of A-CAES is not governed by the Carnot efficiency led to the current 100-200 °C LTA-CAES concept [3].

Figure 3 shows the plant layout of a LTA-CAES. Due to the use of an eight stage radial inflow compressor it is possible to cool the compressed air after each stage. This leads to a reduction of compression work and a limitation of the TES temperature to 100-200 °C. The chosen temperature depends on the economic optimum between increasing revenue through better cycle efficiencies on the one hand and increasing investment costs for the TES due to higher temperatures on the other hand. In the addressed temperature range a pumpable TES medium like pressurized water or thermal oil can be used. Independent of the TES temperature there is always a part of compression heat, which cannot be reused during the discharging process. In the shown stand-alone plant version (Fig. 3) this heat is cooled to ambient air temperature by additional intercoolers. The stored heat is used to pre-heat the compressed air before entering each expander stage. This results in cycle efficiencies of up to 67 %.

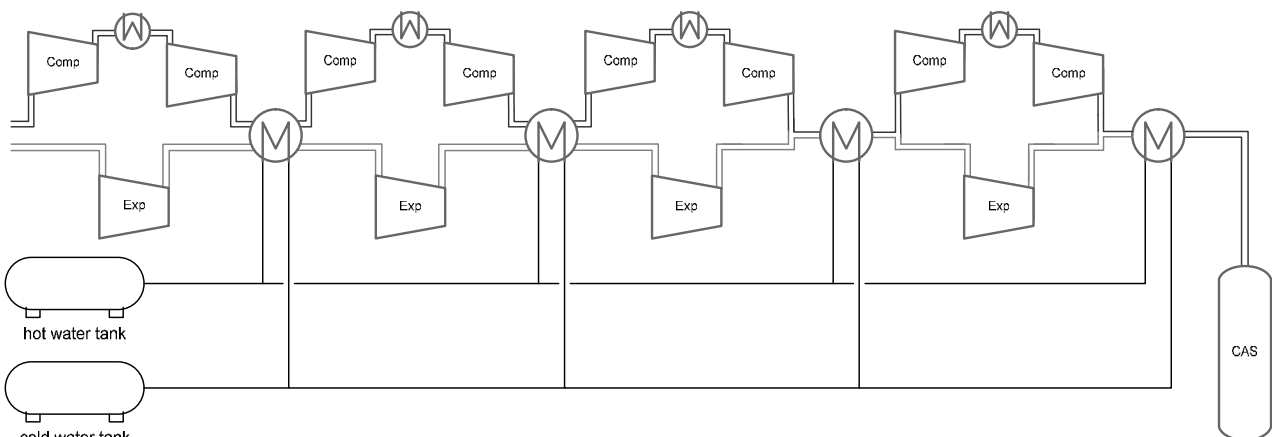


Figure 3: Low-temperature adiabatic compressed air energy storage concept

3 Current plant model

In the ongoing development process the model described below enables the examination of the thermodynamic behavior of the plant especially in off-design operation. The current plant model is based on Modelica 3.2 standard libraries, especially on the Modelica.Fluid library by *Casella et al.* [4], and implemented in Dymola 2012 FD01 [5]. The compressed air is currently assumed as an ideal gas mixture of dry air, taken from the Modelica.Media package ‘DryAirNasa’.

Table 1: Symbol table

symbol	meaning	unit
p	pressure	Pa
h	enthalpy	J/kg
T	temperature	K
X	mass fraction	-
\dot{Q}	heat flow	W
\dot{m}	mass flow	kg/s
y	specific useful flow work	J/kg
κ	isentropic exponent	-
R	gas constant	J/kg K
Π	compression/expansion ratio	-
η	efficiency	-
P	power	W

3.1 Standard library components

As intended by using Modelica to simulate the LTA-CAES, many standard library components could be used in the model. Among sensors, PID-controllers, valves and Fluid.Sources, the whole turbomachinery piping is implemented by Modelica.Fluid pipe models. The CAS is assumed as a solution mined underground salt cavern and is based on the ‘ClosedVolume’ Modelica.Fluid model.

3.2 Heat exchanger modeling

In a first step the used heat exchanger models are simplified to heat sinks and sources without any mass or energy storage capacities. The current approach also assumes an ideally regulated water mass flow through the heat exchanger components.

The pressure loss of the heat exchanger is a fixed value given by the user. The output pressure is calcu-

lated according to Equation 1 if the current mass flow exceeds a given minimum. Otherwise the pressure loss is set to zero. The resulting step is smoothed by a first order transfer function.

$$p_{out} = p_{in} - \Delta p \quad \text{Eq.1}$$

Another fixed input value is the outlet temperature (T_{out}), which is assigned to the air leaving the heat exchangers in times of required heat transfer. Therefore, the enthalpy at the outlet is calculated using this given temperature (Eq. 2).

$$h_{out} = h(p_{out}, T_{out}, X_{i,in}) \quad \text{Eq.2}$$

The required heat flow to reach this temperature is calculated by the energy balance equation (Eq. 3) and given to the user as an output value.

$$0 = \dot{Q}_i + \dot{m}_{in} \cdot h_{in} + \dot{m}_{out} \cdot h_{out} \quad \text{Eq.3}$$

3.3 Turbomachinery modeling

The main components of the LTA-CAES plant are the compressor and expander turbomachineries. Particular attention was paid to these components during model development. The LTA-CAES concept includes an eight stage compressor and a four stage expander, both integrally geared. In the model each of the turbomachinery stages is characterized by two characteristic diagrams, one with regard to the pressure ratio and the other with regard to the polytropic stage efficiency. In the LTA-CAES concept developed so far, both compressor and turbine are supposed to run on fixed speeds.

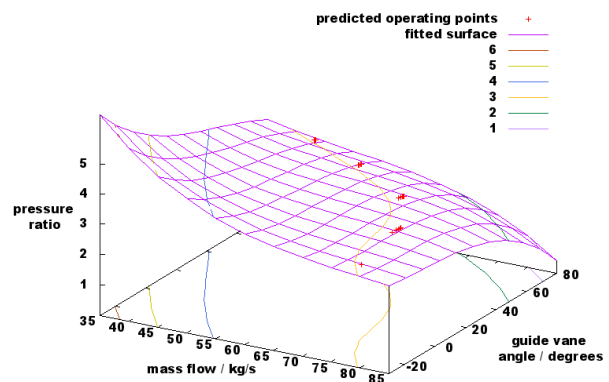


Figure 4: First stage compression ratio diagram

In the LTA-CAES concept the control of the operational point is carried out by variable guide vanes.

Their impact on turbomachinery operation is determined by the current angle of the guide vanes. Figure 4 shows the pressure ratio for the first compressor stage as a function of air mass flow and guide vane angle.

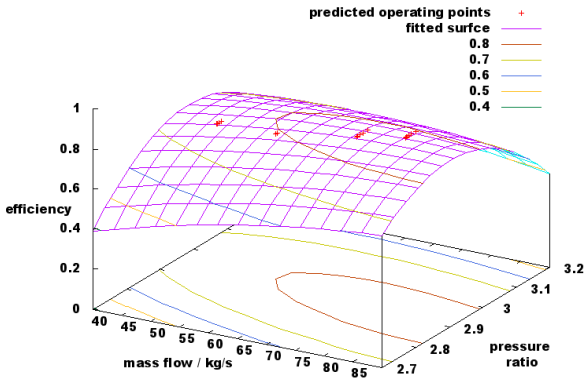


Figure 5: First stage polytropic efficiency diagram

The polytropic efficiency of the same compressor stage is a function of air mass flow and pressure ratio as depicted in Figure 5.

All these diagrams are based on predicted values for operating and off-design points. The parameters of the second (efficiency) and third (compression/expansion ratio) order surface functions are fitted by the open source software GnuPlot in order to reproduce the available off-design point behavior.

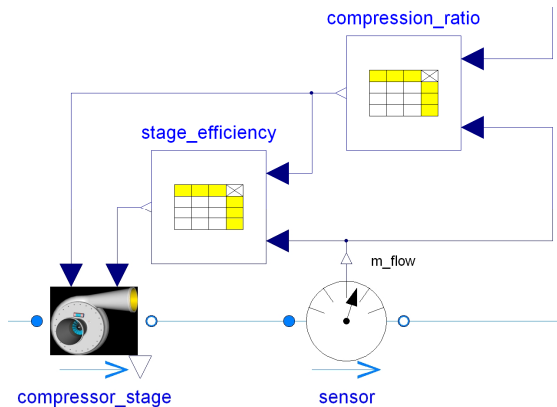


Figure 6: Compressor stage model with inputs from characteristic diagrams

As shown in Figure 6 the characteristic diagrams are implemented as a ‘CombiTable2d’ in Modelica.

With the current mass flow delivered by a sensor and the guide vane angle given by the control instance as inputs, these tables deliver their values as inputs for the Modelica.Fluid based compressor stage model. The model uses these values to calculate the change in enthalpy. Therefore, the isentropic exponent κ is calculated at suction conditions by the used Modelica.Media model. Together with the stage inlet temperature (T_{in}) and the compression ratio input value (Π_{in}), Equation 4 is used to calculate the specific isentropic useful flow work (y_s) assuming ideal gas behavior.

$$y_s = \left(\frac{\kappa}{\kappa - 1} \right) \cdot R \cdot T_{in} \cdot \left[\Pi_{in}^{\frac{\kappa-1}{\kappa}} - 1 \right] \quad \text{Eq.4}$$

Equation 5 shows the conversion of the polytropic efficiency input value ($\eta_{pol, in}$) to isentropic efficiency (η_s).

$$\eta_s = \frac{\Pi_{in}^{\frac{\kappa-1}{\kappa}} - 1}{\Pi_{in}^{\frac{\kappa-1}{\kappa}} \eta_{pol, in} - 1} \quad \text{Eq.5}$$

The division by the isentropic efficiency (η_s) results in the effective change in enthalpy (Δh) (Eq. 6).

$$\Delta h = \frac{y_s}{\eta_s} \quad \text{Eq.6}$$

The output pressure is calculated by Equation 7.

$$p_{out} = p_{in} \cdot \Pi_{in} \quad \text{Eq.7}$$

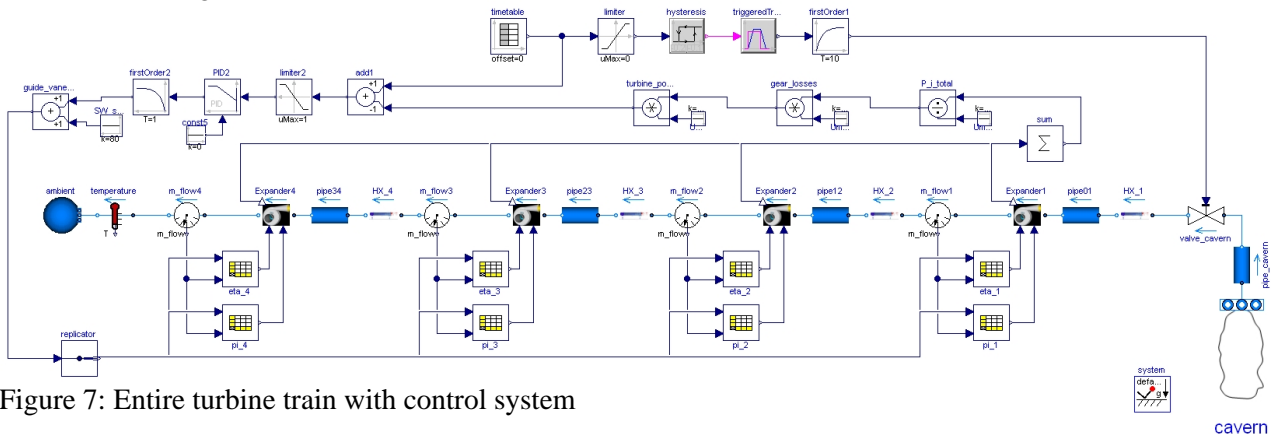


Figure 7: Entire turbine train with control system

With the given pressure and enthalpy at the output port of the compressor stage model, Modelica.Media is able to calculate values like the output temperature.

To match the energy balance (Eq. 8) there is an additional model output value called internal consumed power (P_i).

$$0 = P_i + \dot{m}_{in} \cdot h_{in} + \dot{m}_{out} \cdot h_{out} \quad \text{Eq.8}$$

Since there is no change in air mass flow within the stage, the mass balance shown in Equation 9 is valid.

$$0 = \dot{m}_{in} + \dot{m}_{out} \quad \text{Eq.9}$$

In the turbine model both characteristic diagrams are generated as functions of mass flow and guide vane angle from literature data (Fig. 7) [6, 7]. Apart from that, the same kind of input values and a corresponding set of equations are used. It is therefore not described in detail in this paper.

Simulations of multi-stage turbomachineries are very complex since a pressure or temperature change in one stage has a direct influence on the following ones. This especially applies for off-design conditions and for interim cooling or heating processes. The fundamental advantage of modeling each compressor or turbine stage as an independent model is the ability to calculate their interactions without complex methods like the principle of superposition for the compressor or the law of cone for the turbine.

3.4 Control system

Since the plant model is not intended to simulate start-up and shut-down phase in detail, the modeling of the surge control valve was omitted. The start-up and shut-down of the plant is therefore simplified. Once started, the consumed or produced power is adjusted by a PID-controller system. The values the control system should maintain are given to the model by a Modelica ‘timeTable’ connected to an external file. Programming both stage types in the shown pressure driven way leads to a turbomachinery system with a self-regulating mass flow rate. The control system is able to change the guide vane angle and therefore the operating point of the plant. As proven by comparison, the model behavior and the guide vane angle control simulate the real performance with sufficient accuracy.

4 First results

The presented simulation results refer to a model parameterization of a plant with an eight stage 53 MW compressor unit and a four stage expander unit with an output power of 30 MW. The TES temperature is limited to 150 °C and the cavern pressure varies in the range between 100 and 152 bar assuming a cavern depth of 1500 m. The chosen high pressure leads to a high energy density in the cavern. The required geometrical volume of the cavern is 30,000 m³. With this cavern the LTA-CAES plant is able to operate six full load hours of charging as well as discharging.

4.1 Plant operation

Like normal power plants, usual diabatic CAES plants preferably work only in their full load operating point, where optimal efficiency can be reached. Therefore, the first simulation results show the plant behavior at design point, which reflects a full cycle of charging and discharging the cavern at maximum power.

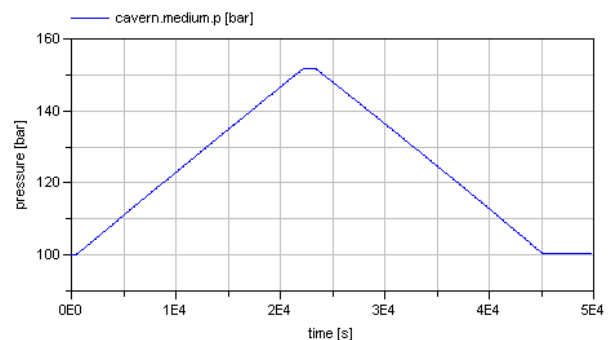


Figure 8: Cavern pressure during full cycle process

Figure 8 shows the cavern pressure increasing from 100 to 152 bar and then decreasing to 100 bar again during a full charge and discharge period of 6 hours (21,600 s) each.

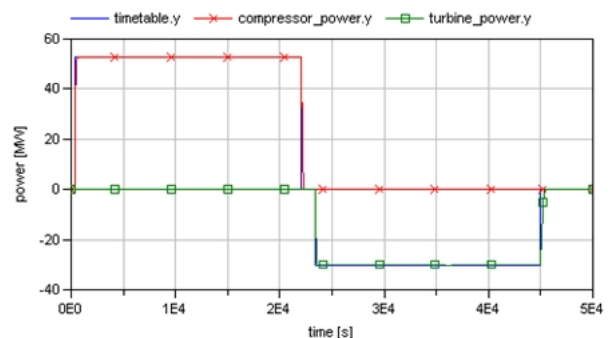


Figure 9: Timetable signal and corresponding compressor power consumption and turbine power output

During the charging process (400-22,000 s) the requested power consumption is set to 53 MW. While discharging the LTA-CAES the requested power output is set to the maximum output power of 30 MW. This call starts at 23,400 s and continues to 45,000 s. As shown in Figure 9 the timetable provides the corresponding values to the model and the compressor and turbine power follows this demand.

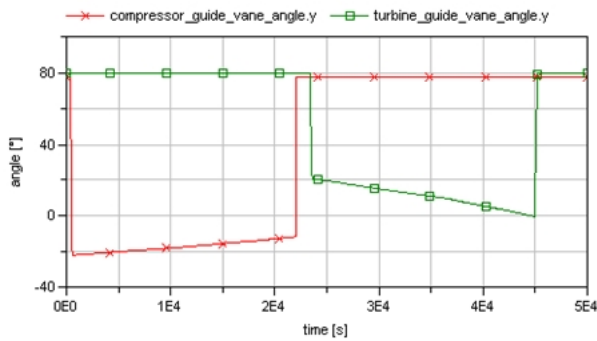


Figure 10: Guide vane angle adjustment

Figure 10 shows the guide vane angle in the same time period, regulated by the control system to match the timetable power signal. It can be seen, that the compressor guide vanes are continuously closing to hold the power consumption of 53 MW during the charging process. In contrast the turbine guide vanes are opened up more and more during the discharge process in order to provide a constant power output of 30 MW. The corresponding change in air mass flow due to the guide vane adjustment is shown in Figure 11.

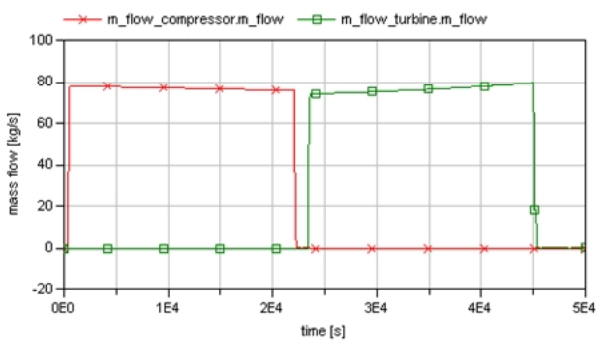


Figure 11: Mass flow through compressor and turbine train

4.2 Compressor train

One aspect of the previously mentioned complex interactions between each of the compressor stages can be observed in Figure 12. It can be seen that the increasing cavern pressure is not reached by a slightly increasing compression ratio in each stage. Rather there is a strong increase of the compression

ratio in the higher stages and nearly no change in the first three stages.

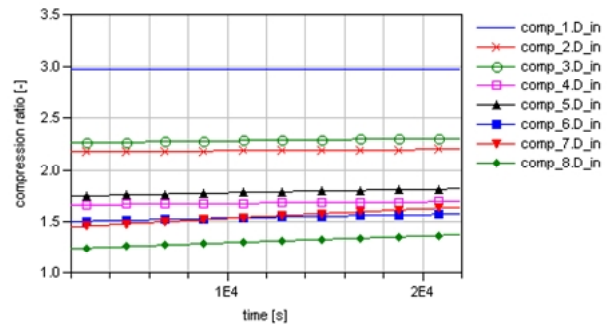


Figure 12: Compressor stages compression ratios

As a result of this compressor behavior the power consumed by the stages does not behave similar (Fig. 13). Especially, the first and the last compressor stage show an opposed development. While all the other stages consume a constant power, due to the reduction of mass flow (Fig. 11) by guide vane adjustment, the last stage consumes more and more power. In this compressor stage the strong increase in compression ratio overcompensates the reduced mass flow. In contrast the power of the first stage decreases.

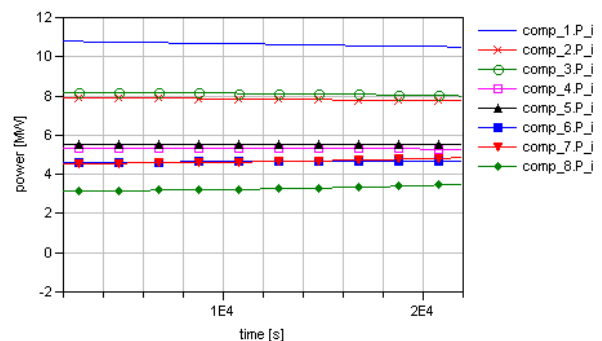


Figure 13: Compressor stages power consumptions

According to the different change of compression ratio and power consumption in all of the stages, the efficiency course also varies between each of them. Depending on that the temperature of the air entering the heat exchangers varies during the charging process. Figure 14 shows the heat flow rates each of the heat exchangers has to provide. The previously described behavior of the stages can be seen here again clearly. The wide range of heat flow rates arises from the different tasks of the heat exchangers. The heat exchangers two, four, six and eight are designed to deliver a preferably constant heat flow to the TES. The others just cool the process air down to a defined temperature to ensure the maximum TES temperature of 150 °C in this plant layout. Therefore, their

heat flows vary according to the compression ratio of the previous compressor stage.

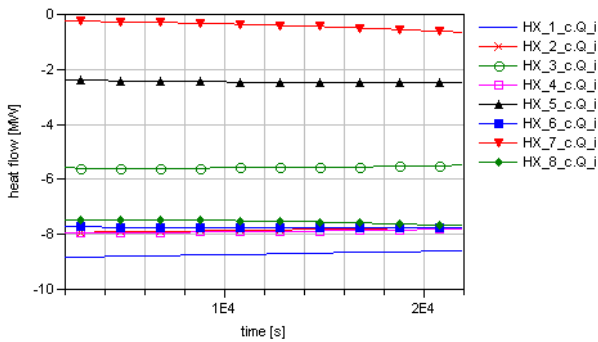


Figure 14: Heat flow rates of the heat exchangers during the compression process

4.3 Turbine train

By opening the turbine guide vanes (Fig. 10) the control system increases the air mass flow (Fig. 11) to compensate the decrease in turbine output power due to the decrease in cavern pressure. Despite the considerable decrease of turbine inlet pressure of 52 bar during discharging, the turbine mass flow increases only by 5.4 kg/s, enough to allow for a constant power output. The corresponding dynamic behavior of each of the four expander stages is shown in the Figures 15 and 16.

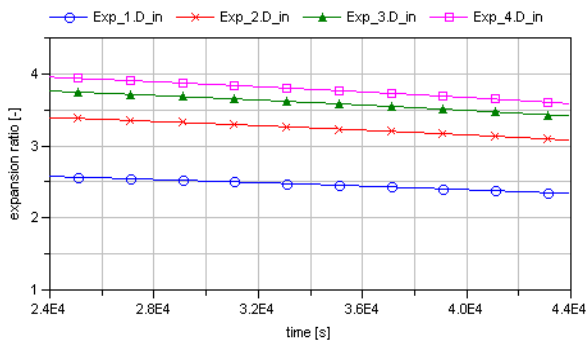


Figure 15: Expander stages expansion ratios

Corresponding to the decreasing cavern pressure depicted in Figure 8, the expansion ratio of each stage decreases as well. It can be seen, that each stage has an individual course (Fig. 15). These pressure driven expansion ratios, together with the adjustable guide vane angle, define the mass flow through the expander stages as depicted in Figure 11.

Depending on the self-adjusting mass flow and the guide vane angle given by the control system, each stage has an individual efficiency course during discharging.

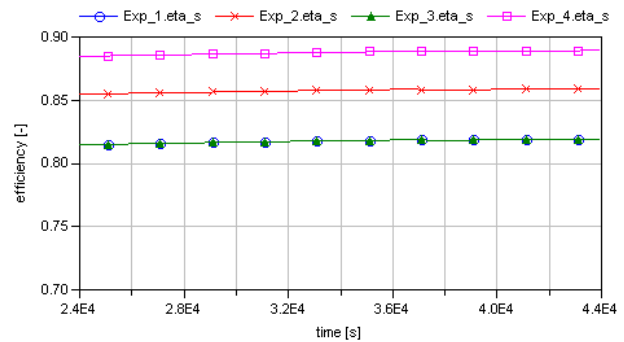


Figure 16: Turbine stages isentropic efficiencies

Figure 16 shows the efficiencies of the four expander stages during the discharging process. The values seem to be constant. In fact there is a minimal rise at the beginning of the discharging process followed by a slightly decrease towards the end. The maximum value can be found at the point the guide vane angle (Fig. 10) crosses the zero degree position, because at this condition the expander stages reach their optimal operating point. The nearly constant efficiency over the whole discharging process demonstrates the general advantage of turbomachinery control by guide vane adjustment.

As a result of the changing expansion ratios and efficiencies the output temperatures of the expander stages vary, too. Because of the small change in efficiencies, the expansion ratios are the main influential variables. Together with the air mass flow these temperatures govern the heat exchanger requirements. Figure 17 shows air temperatures at the inlet of each heat exchanger. Since the assumed slow discharge of the cavern has a negligible effect on the cavern temperature, the air temperature at the first heat exchanger inlet is constantly set to 50 °C. The inlet temperatures for the other heat exchangers are equal to the previous turbine stages output temperatures. As to be expected, increasing temperatures depending on the decreasing expansion ratios of the turbine stages can be observed.

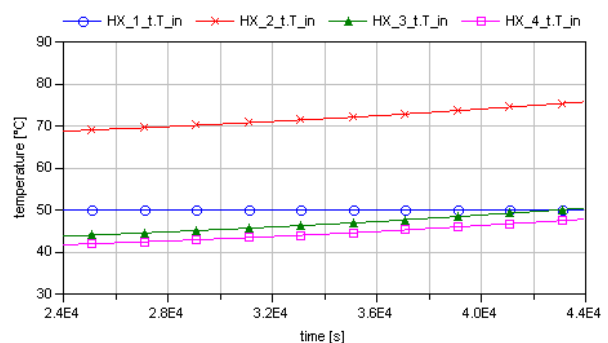


Figure 17: Temperature of the heat exchanger in-flowing air

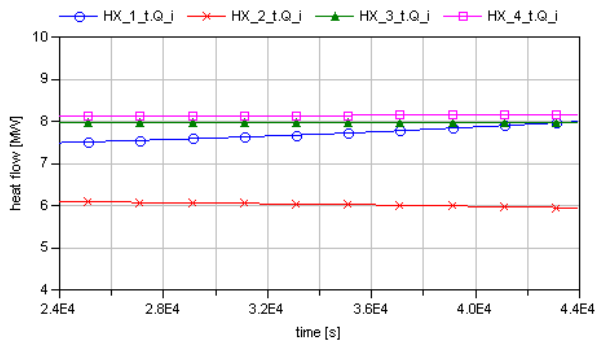


Figure 18: Heat flow rates into the heat exchangers needed to heat up the airflow to 150 °C

The heat flow rates of the heat exchangers to heat up the air to 150 °C show an interesting behavior (Fig. 18). The heat flow rate required by the first heat exchanger is increasing due to the rising mass flow. In contrast the accordingly expected heat flow increase at the other stages is compensated by the higher input temperatures and the consequently smaller temperature difference between inlet and outlet air stream.

4.4 Model performance

The shown results were generated using Dymola 2012 FD01 on a 3 GHz dual-core system with 4 GB of RAM. The model was initialized with initial guess values from stationary calculations. Apart from some difficulties at points of sudden step responses, which could be solved by smoothing these, the model works very fine. The model performs robust and quick, mainly due to neglecting mass and energy storage in compressor and expander stages as well as in the heat exchangers implemented so far. The simulation of the whole 50,000 s charging and discharging cycle (Dassl 0.0001; 500 output intervals) requires 9 s. Leveling up the number of output intervals to 5,000 increases the simulation time to 13.7 s.

5 Conclusions and work in progress

The basic results of the model show the potential of using dynamic simulation to investigate the thermodynamic behavior of an A-CAES. Especially, the complex interactions between turbomachinery stages, heat exchangers and pressure losses can be analyzed in detail. Furthermore, the model allows the analysis of off-design behavior, which is getting more and more important in today's electricity markets. The influence of off-design operation on the overall cycle efficiency can be evaluated as well.

Besides the presented results work further progresses. The presented model will be extended by:

- the implementation of detailed heat exchanger models
- the implementation of external media libraries for humid air to investigate the influence of condensing water in the process
- the implementation of alternative turbomachinery concepts

In the absence of experimental data the compressor model so far had to be validated by data from steady-state calculations in different working points. Here, the model results match the steady-state ones very well. A detailed off-design validation by experimental data for the turbomachinery is aimed at for the future.

The final goal of the research work will be an iterative process by using both, the presented dynamic simulation model and the economic optimization model GOMES[®] [8]. This way an optimization of technical and economical aspects for a given business case will be possible.

6 Acknowledgements

The authors thank the German Federal Ministry for the Environment, Nature Conservation and Nuclear Safety for funding the project »Adiabates Niedertemperaturdruckluftspeicherkraftwerk zur Unterstützung der Netzintegration von Windenergie« (FKZ 0325211).

References

- [1] Wolf D.: Methods for Design and Application of Adiabatic Compressed Air Energy Storage Based on Dynamic Modeling. Bochum, Germany: PhD thesis, Ruhr-Universität Bochum, 2010, UMSICHT-Schriftenreihe 65, urn:nbn:de:0011-n-1546519, 2011.
- [2] Marquardt R., Hoffmann S., Pazzi S., Klafki M., Zunft S.: AA-CAES – Opportunities and challenges of advanced adiabatic compressed air energy storage technology as a balancing tool in interconnected grids. In: 40. Kraftwerkstechnisches Kolloquium 2008, Vol. 2, Technische Universität Dresden (Ed.), 2008.

- [3] Budt M., Wolf D., Prümper H.-J.: A Low-temperature approach to Adiabatic Compressed Air Energy Storage. Proceedings of 12th International Conference on Energy Storage - INNOSTOCK, Lleida, Spain, 2012.
- [4] Casella F., Otter M., Proelss K., Richter Ch., Tummescheit H.: The Modelica Fluid and Media library for modeling of incompressible and compressible thermo-fluid pipe networks. Proceedings of 5th International Modelica Conference, Vienna, Austria, 2006.
- [5] Dassault Systèmes, Dymola 2012 FD01.
- [6] Moustapha H., Zelesky M.F., Baines N.C., Japikse D.: Axial and Radial Turbines. Concepts NREC (Ed.), 2003.
- [7] Bloch H., Soares C.: Turboexpanders and process application. Butterworth-Heinemann (Ed.), 2001.
- [8] Wolf D., Kanngießner A., Budt M., Dötsch Ch.: Adiabatic Compressed Air Energy Storage co-located with wind energy – multifunctional storage commitment optimization for the German market using GOMES. In: Energy Systems, Vol. 3, Issue 2, 2012.

Natural Unit Representation in Modelica

Kevin L. Davies Christiann J.J. Paredis
Georgia Institute of Technology
Atlanta, Georgia USA

Abstract

A method is presented by which alternative systems of physical units may be represented and utilized in Modelica. The method may be useful in simulating models of physical systems where the base units of the International System of Units (Système international d’unités, SI)—the standard unit system in Modelica—are poorly scaled. It also provides a convenient means to express the values of physical quantities in fields of science and engineering where data is typically represented in other systems of units or where the rank of the system of units is less than that of SI (i.e., natural units). By explicitly expressing the value of a physical quantity as the product of a number and a unit (where the unit is an algebraic variable), the method uses variables that are unit-neutral. Unfortunately, workarounds are necessary in order to implement the method in the current version of the Modelica language. Nonetheless, it may be useful in special applications, and the related discussion may provide valuable insight. In particular, it is shown that there is an apparent conflict in the interpretation of “number” and “value” between Modelica and the International Bureau of Weights and Measures (Bureau International des Poids et Mesures, BIPM).

Keywords: natural units; physical quantities; Modelica; SI

1 Introduction

In the mathematical representation of physical systems, the values of quantities are interrelated through equations that express the behavior of the system over time and space. As stated by the BIPM [5, p. 103]:

“The value of a quantity is generally expressed as the product of a number and a unit. The unit is simply a particular example of the quantity concerned which is used as a reference, and the number is the ratio of the value of the quantity to the unit.”

In general, a unit may be the product of powers of other units, whether they are base units or units derived from the base units in the same manner.

In the Modelica language, physical quantities are typically expressed as instances of the `Real` type [12, p. 46]. The `value` attribute of the instance is the number associated with the value of the quantity (not the value of the quantity, as will be seen). The `unit` attribute is a string that describes the unit by which the value of the quantity is divided to arrive at the number.ⁱ The `displayUnit` attribute (also a string) describes the unit by which the value should be divided to arrive at the number as it is entered by or presented to the user. Based on the information provided by the `unit` and `displayUnit` attributes, simulation tools may perform unit checking and conversion. The `Real` type contains other attributes as well, including `quantity`, which is another string [8, p. 375].

The `SIunits` subpackage of the Modelica Standard Library contains types that inherit from the `Real` type. The type definitions appropriately modify the `unit`, `displayUnit`, and `quantity` attributes (among others) to represent various physical quantities. The `unit` and `displayUnit` attributes are based on the SI. The `quantity` string is generally used to describe the name of the physical quantity. For example, the `Velocity` type has a unit of “m/s” and a quantity of “Velocity”.

If an instance of the `Velocity` type has a value of one ($v = 1$), then it is meant that “the value of velocity is equal to one meter per second.” Again, the `value` attribute represents the number, or the value divided by the unit, not the value itself. This apparent conflict could be solved in one of several ways. First, the unit could be strictly set equal to be one (1), regardless of what the unit is. This is the essence of the current implementation in Modelica. It is also the interpretation we use when we are working a problem by hand

ⁱHereafter, the value of the quantity is referred to as simply the value, but it should not be confused with the `value` attribute (which, in the current version of the Modelica language, is the number).

and drop the units because we are exclusively using a particular system of units. However, in this case, the statement that “the value of a quantity is generally expressed as the product of a number and a unit” [5] loses its meaning; it may as well be “the value of a quantity is generally expressed as the number.” Second, the `value` attribute could be renamed as the `number` attribute. Since the name of a variable is an implicit reference to this attribute (whatever it is called), the variable would then represent the number. The third method of resolution is to let the units (the meter and the second in this case) be mathematical entities and let $v' = 1 \cdot \text{m/s}$. Here, the variable v' directly represents the value. Its `value` attribute is the value in the context of the statement by the BIPM.

2 Method

The approach is to follow the third method to resolve the apparent misnomer of the `value` attribute—to factor the units out of the `unit` attribute and into the `value` attribute. This offers the advantage that unit conversion is handled naturally. The essence of unit conversion is that the phrase “ x (value) in u (unit)” is interpreted mathematically as “ x divided by u .” Continuing with the previous example, v' is divided by m/s in order to display v' in meters per second (as a number). The result is simply one (1). If the unit foot is established through the appropriate relation ($\text{ft} \approx 0.3048 \cdot \text{m}$) and v' is divided by ft/s , the result is v' in feet per second (~ 3.2894).

As another example, systems involving angle are sometimes evaluated by working with variables in cycles and other times with variables in radians. If the variable is the value, then “variable in unit” means “value divided by unit.” If we work with the value directly, then there is no need to specify which unit we are working “in.” The unit is included; it has not been factored out by division. As long as the dimensionality is correct, the math is equivalent due to the relationships among units (or combinations of units). In this case, the relevant unit relation is $1 \cdot \text{cycle} = 2\pi \cdot \text{rad}$.ⁱⁱ This example extends directly to frequency (angle per time). Often, different symbols are used for frequency in Hz (ν) and frequency in rad/s (ω). If the units are included in the variable f , then $f = \nu \cdot \text{Hz} = \omega \cdot \text{rad/s}$.

In this method, each unit must be represented by an

ⁱⁱFurthermore, a cycle is typically equated to the number one (1). For instance, in SI, a frequency of one hertz ($1 \cdot \text{Hz}$) is equated to one per second ($1/\text{s}$) [5] even though to be precise it is one cycle per second ($1 \cdot \text{cycle/s}$).

algebraic variable (albeit constant). For each unit introduced, it is necessary to add an equation that allows the unit’s value to be determined. If a unit is considered to be a derived unit, then the equation simply relates the unit to other units (e.g., $1 \cdot \text{cycle} = 2\pi \cdot \text{rad}$). However, there are several units (in SI, 7) that may not be simply defined via other units. These base units must be related to something outside of the algebraic system of equations representing the immediate physical system. This something is the “particular example of the quantity concerned which is used as a reference” quoted previously [5]. The designation of “base” or “derived” is somewhat arbitrary [8, p. 375], but regardless, there are a number of units that must be defined by example. Considering only the immediate physical system, these units are linearly independent.

If only the SI will be used, then it is easiest to strictly set each of the base units of SI equal to one (1)—the meter (m), kilogram (kg), second (s), ampere (A), kelvin (K), mole (mol), and candela (cd). This is implicitly the case in `Modelica.SIunits`, but again, it hardly captures the idea that a value is the product of a number and a unit.

There are systems where typical values are many orders of magnitude larger or smaller than the related product of powers of base SI units (e.g., the domains of astrophysics and atomic physics). In modeling and simulating those systems, it may be advantageous to choose appropriately small or large values (respectively) for the corresponding base units such that the product of the number (large or small in magnitude) and the unit (small or large, respectively) is well-scaled. Products of this type are often involved in initial conditions or parameter expressions, which are not time-varying. Therefore, the number and the unit can be multiplied before the dynamic simulation. During the simulation, only the value is important. After the simulation, the trajectory of the value may be divided by the unit for display. This scaling is usually unnecessary due to the wide range and appropriate distribution of the real numbers that are representable in floating point space. The Modelica language specification recommends that floating point numbers be represented in at least IEEE double precision, which covers magnitudes from $\sim 2.225 \times 10^{-308}$ to $\sim 1.798 \times 10^{308}$ [12, p. 13]. However, in some cases it may be preferable to carefully scale the units and use single precision instead for the sake of computational performance. There are fields of research where, even today, simulations are sometimes performed in single precision [10] and where scaling is a concern [14, p. 29].

Since there are many systems of units besides the SI, it is best if the method is neutral with regards to not only the values of the base units, but also the choice of the base units and even the number of base units. As mentioned previously, the choice of base units is somewhat arbitrary, and different systems of units are based on different choices. Some systems of units have fewer base units (lower rank) than SI, since additional constraints are added that exchange base units for derived units. For example, the Planck, Stoney, Hartree, and Ryberg systems of units define the Boltzmann constant to be equal to one ($k = 1$) [15]. The unit K is “eliminated” [9, p. 386] or, more precisely, considered a derived unit instead of a base unit. In the SI, the Boltzmann constant would be derived from the base units kilogram, meter, and second ($K \approx 1.381 \times 10^{-23} \cdot \text{kg} \cdot \text{m}^2/\text{s}^2$). In such a system, terms that would otherwise be written as kT may be replaced by simply T ; temperature (T) is considered to be energy per particle or degree of freedom. In this case, it is not possible to arbitrarily choose a value for K.

A unit is considered to be a “natural” unit if it depends only on values of universal physical constants [15]. If a system of units is purely natural, then all its base “units” are base “constants.” The “particular example of the quantity concerned which is used as a reference” [5] is an experiment that yields precise and repeatable results in determining a constant rather than a prototype which is carefully controlled and distributed via replicas. For instance, a natural unit for electrical resistance is the von Klitzing constant, and it can be chosen as a base constant. Often, the base constants are defined to be equal to one. However, just as it is not necessary to set base units to one, it is not necessary to set base constants to one. The values can be chosen to best scale the numerics of the system.

It is judicious to check that the terms of each equation have the same dimension. Fortunately, methods for unit checking have already been established and implemented in Dymola [11]. In the present context, those methods can, in theory, be applied to the dimension instead (i.e., “dimension checking” instead of “unit checking”). Again, in the present method, the unit is included in the `value` attribute. The question of which unit the variable is “in” is not applicable, but it is still possible and appropriate to check the dimensions.

The dimension of a value may be expressed in the same manner as the unit is in the current version of the Modelica language [12, Ch. 18]. For SI, it would

be appropriate to use these base dimensions instead of the corresponding base units: length (L), mass (M), time (T), electric current (I), thermodynamic temperature (Theta), amount of substance (N), and luminous intensity (J) [5, p. 105]. In the example that follows, the Rydberg constant, Faraday constant, and the specific mass of electrons are all set equal to one. Therefore, the rank is reduced from seven to four.

3 Implementation

The method is implemented in version 3.2 of the Modelica language [12] and version 7.4 of Dymola [7]. However, the implementation includes several less-than-ideal workarounds; a full and consistent implementation would require changes to the language and the modeling environment (see Sec. 4).

First, it is necessary to define the units and constants as variables. These variables must be declared in an accessible package so that they can be used in equations within the declaration, initial, and dynamic sections of the model and its subclasses. An excerpt from this `Units` package is shown in Listing 1. The top section of the code establishes mathematical constants (in this case, only π). The next section establishes the base constants and units, which are adjustable. The third section establishes the constants and units which may be derived from the base units and constants using accepted empirical relations. The rest of the code (not listed) establishes the SI prefixes and the remaining derived units and constants. The SI prefixes are included in their unabbreviated form in order to avoid name conflicts (e.g., `constant Real kilo(unit="1")=1E3`). In a model, a kilometer is included as `kilo*m`, unless `km` is defined as a stand-alone unit. All of the primary units of SI are included (Tables 1 and 3 of [5]) except for $^{\circ}\text{C}$, since it involves an offset. Other convenient units are included for the system at hand (e.g., `atm`). For convenience, the `Units` package is given the abbreviated label `U` by an `import` statement at the top level of the entire library or containing package.

Each unit or constant is a `constant Real`. The `unit` attribute is given a string that describes the dimension. The abbreviations `l`, `N`, `T`, and `I` are used for length, number, time, and luminous intensity, respectively.ⁱⁱⁱ The dimensions are combined as strings

ⁱⁱⁱLowercase “ell” is used so that Dymola 7.4 recognizes it as a unit—the liter. Dymola also recognizes `N` as newton and `T` as tesla. This is not the meaning here, but there is no problem since it happens that these three units are orthogonal. As long as lu-

according to the rules established for unit strings in the Modelica language [12, p. 210].

The units, constants, and prefixes must be identically defined in Dymola's workspace so that they can be used to convert values to numbers for display. The definitions from the Units package are copied to a Modelica script. All the specifications of `constant Real` and of the unit attribute are removed. It is important that the base units or constants are declared at the beginning of the script and all derived units are arranged in an order that allows the script to succeed on the first pass. The script is run when Dymola is launched. Assert statements are added at the end of the script to perform basic checks on the relationships among the values.

Now, types must be defined for the required quantities. Each quantity inherits from the `Real` type. The unit attribute is given a string that describes the dimension (as in the Units package). The quantity attribute is not used, since the type is the quantity. The `displayUnit` attribute is given a string that describes the desired unit to be used for display (according to the format specified in Ch. 18 of [12]). By default, it is the simplest expression of the unit in SI. For convenience, the package containing the quantities is given the global, abbreviated label Q.

Another Modelica script is written to define the unit conversions for display using Dymola's `defineUnitConversion` command. As mentioned previously, a value is divided by a unit to arrive at a number for display. This script is executed after the script that defines the units, constants, and prefixes (automatically—upon starting Dymola) so that all of those variables are available. For example, the entry for velocity is `defineUnitConversion("l/T", "m/s", s/m)`.

A top-level “environment” model is included which stores copies of the base units or constants. With that information, it is possible to re-derive all of the other units and constants. This is important in order to properly interpret simulation results even after the base units or constants are re-adjusted.

Where the `der` operator is used, it is explicitly divided by the unit second (e.g., `der(x)/U.s`). This is necessary because the global variable `time` is time in seconds.

Listing 1: Selected constants from the Units package

```
// -----
//
// minus intensity is not represented in the model (I, which is not
// recognized), unit checking may be used as dimension checking.
```

```
// Base physical constants and units
replaceable constant Bases.Default base
  constrainedby Bases.Basis
"Scaleable base constants and units";
// Note: The base constants and units may be
// replaced to suit the scale of the physical
// system.

final constant Q.Angle rad=base.rad "radian";
final constant Q.Wavenumber R_inf=base.R_inf
  "Rydberg constant (R_&infin;)";
final constant Q.Velocity c=base.c
  "speed of light in vacuum (c)";
final constant Q.MagneticFluxReciprocal k_J=
  base.k_J
  "Josephson constant (k_J)";
final constant Q.Resistance R_K=base.R_K
  "von Klitzing constant (R_K)";
final constant Q.RadiantIntensity 'cd'=base.'cd' "
  candela";
final constant Q.Number k_F=base.k_F
  "Faraday constant (k_F)";
final constant Q.Number R=base.R "gas constant";

// -----
// Empirical constants and units
// Note: The values are currently based on the
// those from NIST (2010). The measured (rather
// than conventional) values are used.

constant Q.Length m=10973731.568539*rad/R_inf "
  meter";
// SI unit of length
// This is the "Rydberg constant" relation (NIST,
// 2010). The unit radian is included to be
// explicit, although it is currently one by
// definition (BIPM, 2006).
// (http://en.wikipedia.org/wiki/Rydberg\_constant)
constant Q.Time s=299792458*m/c "second";
// SI unit of time or duration
// This is the "speed of light in vacuum" relation
// (NIST, 2010).
constant Q.MagneticFlux Wb=483597.870E9/k_J "weber
  ";
// SI unit of magnetic flux
// This is the "Josephson constant" relation
// (NIST, 2010).
constant Q.Conductance S=25812.8074434/R_K "siemen
  ";
// SI unit of electrical conductance
// This is the "von Klitzing constant" relation
// (NIST, 2010). The unit radian is included on
// the denominator for dimensional consistency,
// but it is one by the current definition (BIPM,
// 2006).
constant Q.ParticleNumber mol=96485.3365*Wb*S/k_F
  "mole";
// SI unit of amount of substance
// This is the "Faraday constant" relation (NIST,
// 2010). The factor Wb*S is the coulomb, which
// is defined below.
```

```

constant Q.Potential K=8.3144621*(Wb*rad)^2*S/(s*
  mol*R) "kelvin";
// This is the "molar gas constant" relation
// (NIST, 2010). The factor (Wb*rad)^2*S/s is the
// joule, which is defined below.

```

Listing 2: Selected records from the `Units.Bases` package

```

record Basis "Base constants and units"

  final constant Q.Angle rad=1 "radian";
  // SI unit of rotation or planar angle
  constant Q.Wavenumber R_inf=1
    "Rydberg constant (R_&infin;)";
  // The SI unit length (meter) is inversely
  // proportional to this value, which should be
  // increased for larger characteristic lengths.
  constant Q.Velocity c=1 "speed of light in
    vacuum (c)";
  // The SI unit time (second) is inversely
  // proportional to this value (and R_inf), which
  // should be increased for larger characteristic
  // times.
  constant Q.MagneticFluxReciprocal k_J=1
    "Josephson constant (k_J)";
  // The SI unit of magnetic flux (weber) is
  // inversely proportional to this value, which
  // should be increased for larger magnetic flux
  // numbers. Also, the SI unit of charge
  // (coulomb) is inversely proportional to this
  // value.
  constant Q.Resistance R_K=1
    "von Klitzing constant (R_K)";
  // The SI unit of electrical conductance
  // (siemen) is inversely proportional to this
  // value, which should be increased for larger
  // characteristic conductances. Also, the SI
  // unit of charge (coulomb) is inversely
  // proportional to this value.
  constant Q.RadiantIntensity 'cd'=1 "candela";
  // SI unit of luminous intensity
  constant Q.Number k_F=1 "Faraday constant (k_F)"
    ;
  // The unit of substance (mole) is inversely
  // proportional to this value, which should be
  // increased for larger particle numbers. If
  // k_F is set to 1, then charge is considered
  // to be an amount of substance.
  constant Q.Number R=1 "gas constant";
  // The unit of temperature (kelvin) is inversely
  // proportional to this value, which should be
  // increased for larger temperature numbers. If
  // R is set to 1, then temperature is
  // considered to be a potential.
end Basis;

record Am
  "Base constants and units for SI with k_F and R
  normalized instead of A and m"

  extends Basis(
    final R_inf=sqrt(8.3144621)*10973731.568539,

```

```

    final c=299792458/sqrt(8.3144621),
    final R_K=(96485.3365^2*25812.8074434)/8
      .3144621,
    final k_J=483597.870E9*sqrt(S*s)/m,
    final candela=1,
    final k_F=1,
    final R=1);
  // Note: The values of the un-normalized SI
  // base units are:
  //   A ~ = 0.0000103643
  //   m ~ = 0.346803
end Am;

```

4 Discussion and Conclusion

The implementation has been utilized to help model and simulate a proton exchange membrane fuel cell (PEMFC) in Dymola 7.4 [6]. It has been convenient in specifying the values of parameters and constants in this domain, where the product and research literature quotes values according to many different conventions. There are also cases where simulations have failed until the base constants were adjusted to properly scale critical values. In these cases, adjusting the nominal attributes of the variables did not seem to be sufficient, although it is difficult to prove.

The implementation raises the following concerns, which must be addressed in order to fully and consistently employ the method.

1. The unit attribute of a `Real` type should be renamed as `dimension` to indicate that it represents the physical dimension of the quantity rather than a particular unit.
2. In the new context, the `Real` type may be a misnomer. It may be best renamed as `Quantity`, but this may have implications on the name for the `Complex` record described in the Modelica language specification [12].
3. The quantity attribute of the `Real` type (possibly renamed as `Quantity`) may be superfluous. However, its removal may imply that the same attribute of the `Boolean`, `Integer`, and `String` types should be removed as well.
4. It would be helpful to establish a standard method to store and access the values of the base units and constants along with the results of a simulation. Ideally, the conversions created by the `defineUnitConversion` command (in Dymola) would be dynamically linked to the values of

the base units or constants, regardless of whether they are within an active model or from previous results.

5. The global variable `time` should be expressed as a quantity in the same manner as other variables—as the product of a number and a unit. Currently, `time` is time in unit seconds and the second has a value of 1. The `time` variable should be adjusted such that `time/U.s` is time in unit seconds and the second is not constrained to the value of 1. If the `der` operator is based on this unit-neutral time quantity, then it would be unnecessary to divide its output by the unit second (as in Sec. 3).

All of these items would affect both the Modelica language and the Modelica Standard Library. Therefore, it would be a rather significant undertaking to implement the method as a standard. However, not all of the items are necessary and the method can already be implemented to a limited extent (with work-arounds) in Modelica 3.2 and Dymola 7.4.

If a generalized method of units were to be introduced to Modelica, concepts from SysML may be pertinent and useful. Subsections C.4 and C.5 of version 1.2 of the SysML specification describe model libraries for “Quantity Kinds and Units” and “Quantities, Units, Dimensions, and Values” [1].

The proposed approach is not intended to supersede the previous work in unit checking in Modelica by Broman et al. ([4, 3]). Instead, it uses the methods of unit checking for dimension checking.

Acknowledgments

The authors wish to acknowledge support from the Presidential Fellowship of the Georgia Institute of Technology and the Robert G. Shackelford Fellowship of the Georgia Tech Research Institute.

References

- [1] OMG Systems Modeling Language (OMG SysML[®]), Jun. 2010. Ver. 1.2.
- [2] E. Allen, D. Chase, V. Luchangco, J.-W. Maessen, and G. L. S. Jr. Object-oriented units of measurement. In *OOPSLA04*, Vancouver, BC, Canada, Oct. 2004. ACM 1-58113-712-5/03/0010.
- [3] P. Aronsson and D. Broman. Extendable physical unit checking with understandable error reporting. In *Proc. 7th Int. Modelica Conf.*, Como, Italy, Sep. 2009. Modelica Association.
- [4] D. Broman, P. Aronsson, and P. Fritzson. Design considerations for dimensional inference and unit consistency checking in Modelica. In *Proc. 6th Int. Modelica Conf.*, Bielefeld, Germany, Mar. 2008. Modelica Association.
- [5] Bureau International des Poids et Mesures. The International System of Units (SI). http://www.bipm.org/en/si/si_brochure/, Mar. 2006.
- [6] K. L. Davies, C. J. Paredis, and C. L. Haynes. Library for first-principle models of proton exchange membrane fuel cells in Modelica. In *Proc. 9th Int. Modelica Conf.*, Munich, Germany, Sep. 2012 (accepted). Modelica Assoc.
- [7] Dynasim AB. Dymola: Dynamic Modeling Laboratory, Mar. 2010. Ver. 7.4.
- [8] P. Fritzson. *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*. IEEE Press, Piscataway, NJ, 2004.
- [9] W. Greiner, L. Neise, and H. Stöcker. *Thermodynamics and statistical mechanics*. Classical theoretical physics. Springer-Verlag, 1995.
- [10] B. Hess, C. Kutzner, D. van der Spoel, and E. Lindahl. Gromacs 4: Algorithms for highly efficient, load-balanced, and scalable molecular simulation. *J. Chem. Theory Comput.*, 4(3):435–447, 2008.
- [11] S. E. Mattsson and H. Elmqvist. Unit checking and quantity conservation. In *Proc. 6th Int. Modelica Conf.*, University of Applied Sciences, Bielefeld, Germany, Mar. 2008. Modelica Assoc.
- [12] Modelica Assoc. Modelica: A unified object-oriented language for physical systems modeling: Language specification. <https://www.modelica.org/documents/ModelicaSpec32.pdf>, Mar. 2010. Ver. 3.2.
- [13] National Institute of Science and Technology. Fundamental physical constants—complete listing. <http://physics.nist.gov/cuu/Constants/Table/allascii.txt>, 2010. Accessed Jun. 2012.

- [14] D. C. Rapaport. *The Art of Molecular Dynamics Simulation*. Cambridge University Press, 2nd edition, Apr. 2004.
- [15] Wikipedia. Natural units. http://en.wikipedia.org/wiki/Natural_units. Accessed Mar. 2012.

Modelica Code Generation with Polymorphic Arrays and Records Used in Wind Turbine Modeling

Roland Samlaus¹ Peter Fritzson² Adam Zuga¹ Michael Strobel¹ Claudio Hillmann¹
Fraunhofer Institute for Wind Energy and Energy System Technology IWES¹
Linköpings universitet, Dept. of Computer and Information Science²

Abstract

At Fraunhofer Institute for Wind Energy and Energy System Technology IWES a simulation software for offshore wind farms is being developed, concentrating on the ability to define physical models at different levels of detail. Therefore parameterizable models representing parts of wind turbines are defined that can be transformed for various purposes like simulation with Finite Element Method (FEM) tools or Modelica solvers.

This paper describes the concepts of purely parametric physical models and code generation. It is elucidated how models of different complexity can be transformed into each other by model driven development techniques. Thereby the focus is set on the generation of Modelica code and it is explained how the use of Modelica libraries simplifies the generation of simulatable code.

During the development of generators for Modelica, issues arose regarding type compatibility of arrays with different sizes when using polymorphism. These issues are explained by an example and possible enhancements for the Modelica language are suggested.

Keywords: model transformation; polymorphism; code generation; wind turbine modeling

1 Introduction

At Fraunhofer Institute for Wind Energy and Energy System Technology IWES a simulation software for offshore wind farms is being developed under the project name OneWind. The goal is to provide a tool that allows wind turbine designers and manufacturers to rapidly develop models of wind turbines in different levels of detail. It shall also be possible to use different types of models and to transform them into each other in order to check the models against the users expectations with the best suitable simulation technique. Furthermore, simulations of different load cases ac-

ording to the respective wind turbine standards and guidelines [5, 3] will be possible. A key purpose of the OneWind project is to implement the load calculation as a coupled aero-servo-hydro-elastic simulation in Modelica, to get a better estimation of the turbine performance, to analyze the system response and to optimize the component and control system design.

Nowadays many tools are involved in the process of wind turbine design like GH Bladed¹ for load calculations or Focus² for rotorblade designs, just to name two of them. Additionally Computational Fluid Dynamics (CFD) tools give a more precise view on aerodynamical influences from 3-D flow effects on rotating blades. All of these tools define their own data and model representations and hence provide only limited interoperability. The OneWind project aims to provide consistency in the highly iterative design process between different models for various purposes at design time. Therefore it facilitates the usability of the tools in one integrated development process by introducing a purely parametric data layer called Engineer Design Data (EDD) [17]. Hence data from external tools must be imported into the parametric representation and reverse transformations to the tools data model must be performed in order to generate compatible data as input for simulations with the external tools. Figure 1 displays the concept of the EDD with transformations and code generation.

Due to the different domains that the simulation environments are aiming at, there can not be just one model in the parametric level that represents all kinds of physical properties of a wind turbine component. As an example, structural models of rotorblades must be fairly simple with only few degrees of freedom (e.g. a modal description) in order to be able to execute load calculation for thousands of loadcases in a reasonable time. In contrast, the detailed design of the

¹<http://www.gl-garradhassan.com/en/GHBladed.php>

²<http://www.wmc.eu/focus6.php>

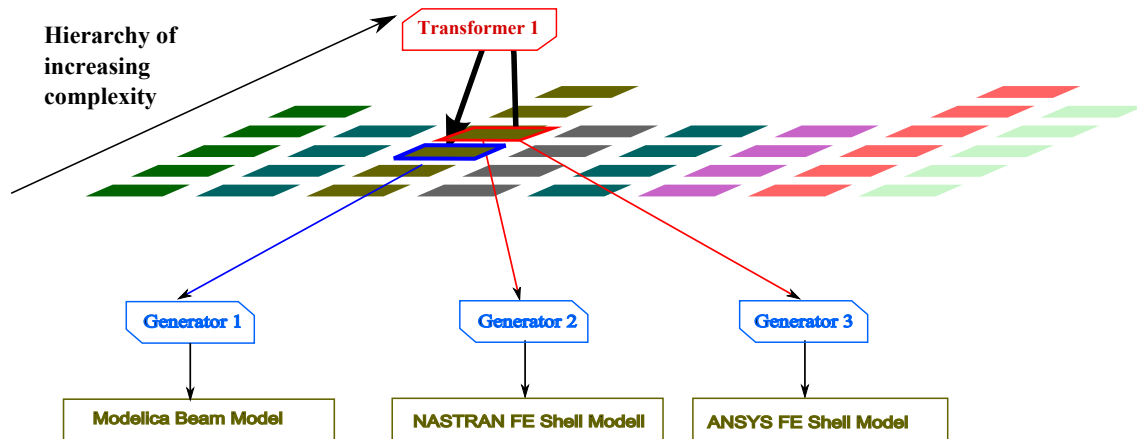


Figure 1: Example of a Transformation Between Two Types of Models and Code Generation for Simulation

composite structure of a rotorblade needs a model with fine grained information about the layer structure to be used for Finite Element Method (FEM) simulations. During the design process changes in the fine-grained models need to be transferred in each iteration step to the simpler models of the load calculation. These transformations from fine to coarse-grained models can often be done automatically. The opposite transformation direction is called design transformation and needs additional user inputs and engineering know how in order to be performed. However, the details of these transformations and their underlying theories are out of scope of this paper. More of interest is the transformation from the EDD representation of wind turbine models to computable Modelica models, as one feature of the OneWind development environment. In the domain of wind turbine modeling with Modelica we can use the EDD to reduce the complexity of model parameterization for the user. Instead of editing the potentially complex source code directly, the user only sees the model parameters that are crucial for the model's behaviour. Thus, the user does not need to understand the syntax of Modelica. Instead of transforming the complete model to a Modelica representation, only the user-defined parameters are transformed to Modelica records, that belong to components of the OneWind Modelica library [18]. The library consists of major components in different complexity levels needed for load calculations of typical offshore wind turbines. Components for the structure and aerodynamics of rotor blades are provided as well as a hub, nacelle with drivetrain and generator, tower, substructure and operating control procedures. Additionally, the library includes models for the simulation of external conditions, (wind, soil and waves) and their influence on the wind turbine's structure. The library

is constructed in a way, that the assembly of model classes with related parameter classes can be manipulated by redeclaration statements and inheritance from the base library classes. An example is shown in section 4. This has the benefit that developers of Modelica models can re-use the components of the library, individually change parameters of the model classes and easily enhance it. Furthermore a wind turbine model with a desired complexity level can be constructed using a custom combination of library models.

The remainder of this paper is structured as follows: In section 2, the concept of the EDD is introduced concentrating on a simple wind turbine model. In section 3 the transformation from the EDD to the Modelica representation is explained. Section 4 elucidates the problems that we encountered by transforming user defined parameters to a Modelica array representation. In section 5, we finally come to a conclusion and suggest how polymorphism of the Modelica language could be enhanced by introducing polymorphism in arrays.

2 Engineer Design Data

The concept of a purely parametric data layer is used in all products of the OneWind project. It represents the idea to ensure the consistency of models in different levels of detail for all purposes needed during the design process of a wind turbine. In this layer the user can manipulate models which are imported or newly created in a unified way, regardless of the software used for further processing or simulation. The models can then be transformed to a computable form and simulated by external tools. When the simulation results are obtained, the user can analyze and assess the results and start with a new design iteration in order to

enhance the physical models. This section introduces how EDD models with different representation types can be defined.

2.1 Abstract Syntax Definition

A meta-model hierarchy called Meta-Object Facility (MOF) [15] for the definition of models is defined by the Object Management Group (OMG) (see Figure 2). The hierarchy is specified as follows: The M0 level describes objects of the real world, as an example it could be an instance of a model of a wind turbine with specific parameters. The model layer (M1) defines, how a real life object can be represented, e.g. by defining a wind turbine model with Modelica. The model consists of components like rotorblades, a tower and a hub that are represented as class or model definitions. In the meta level (M2) the objects that can be used for the development of M1 models are described. For Modelica this implies, that the different language constructs like classes, models, equations, ... are defined. Finally the meta-meta level (M3) describes, how the M2 models are defined. We selected Eclipse [2] as the base environment for our products since the Integrated Development Environment (IDE) is open source and can be customized easily by plug-ins that are implemented by software developers. Since the underlying framework is Eclipse, we picked the Eclipse Modeling Framework (EMF) [1] as the meta meta-model for our Modelica language definition as well as the parametric data layer. Using EMF as the M3 layer implementation, language constructs like Modelica classes and equations are defined by EClasses, type references from a component to model declarations can be defined with EReferences and so on. EMF implements a basic version of the MOF, called EMOF. Hence the parametric layer used in OneWind is build by various meta-model definitions defined with EMF. EDD models specified in the meta-models implement a common interface. One strength of the meta-model approach is that one can define automatic transformations between two models of the same level, if one can define relations between features on the meta level. EMF is widely used in the Eclipse community and hence many tools exist that simplify the definition and use of the models. The Framework allows generic processing of model instances making it possible to create functionality for a wide spectrum of diverse models. As an example we implemented a generic editor enabling the user to edit arbitrary models that are based on an EMF meta-model. Generic SWT-composites for basic data types like `double`, `int` and `String` are avail-

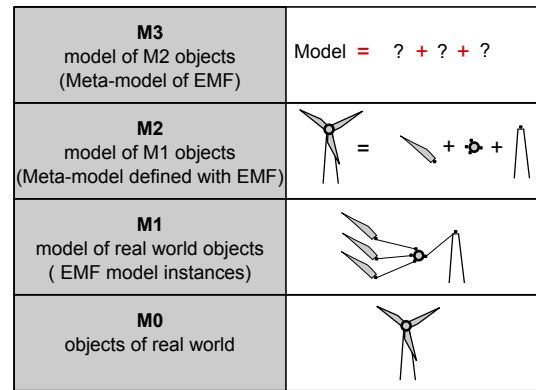


Figure 2: Hierarchy of Meta-models Defined by the OMG

able. Special composites can be easily registered as an OSGi [13] service in order to provide a convenient way to edit custom data types.

2.2 Concrete Syntax Definition

The abstract syntax of a Domain-Specific Language (DSL) is described by its meta-model, i.e., it defines how the language is logically structured. A stored model definition in the Modelica language for example is defined as the root element of a document, which can be a source file containing the model starting with an optional `within` statement that defines in which package the model is contained. Inside the stored definition, packages, classes, sub-classes etc. are defined according to the rules of the abstract syntax.

The Modelica specification [14] defines an accompanying concrete syntax grammar for textual representation of the language. Parsers use the grammar definition to recognize the elements of the language and to build a tree based representation of Modelica models. In our project the textual DSL Modelica is defined with Xtext [12], allowing us to describe textual representations and to automatically generate EMF-based meta models. Through the grammar definition textual editors that developers can use to define models using the language are automatically generated. The editors recognize syntax errors and the IDE can check additional Well-Formedness Rules (WFR) on the user defined models [16] in order to assist the developer in creating correct code.

A textual representation of DSLs is one way of model representation. A graphical notation may also be defined consisting of icons for structural features. A popular example is the graphical notation language Unified Modeling Language (UML) [6]. It defines icons like rectangular boxes for classes or lines be-

tween classes representing associations. When using EMF for the definition of meta-models, it is not necessary to define a graphical representation. The framework provides generic tree-based editors for editing model instances. The standard serialization is based on the XMI file format. However, tools like the Graphical Modeling Framework (GMF) [8] allow the definition of graphical notations similar to the previously mentioned UML for custom meta-models defined with EMF. Hence we have three types of representations for DSLs:

1. Abstract syntax without graphical notation
2. Abstract syntax with textual concrete syntax
3. Abstract syntax with icon based concrete syntax

As we have seen, there are multiple ways of representing EMF based meta-models. Hence it is possible to use the appropriate way of representation for each kind of data. The data can still be processed in a similar way since the underlying data model is the same. In the OneWind project we use the first form of DSLs without graphical notation for the parameterizable components of our wind turbine, i.e., for EDD models. Besides the tree based editor that is provided by EMF we implemented a more convenient and extensible editor based on SWT composites as mentioned above. The textual DSLs currently supported are Modelica, the data format ANSYS Parametric Design Language (APDL)³ and a definition language for airfoils. Xtext grammars were defined for these formats resulting in generated editors, parsers and serializers (also known as unparsers). Since Xtext uses AntLR for the parser generation, support for some formats like NASTRAN⁴ bulk data format are hard to implement.

Pure data formats are often structured by terminal symbols like white spaces or line breaks that complicate or even make it impossible to define a LL(k)-grammar [11]. Hence, a custom parser and serializer/unparser could be implemented in the future that would create an EMF compatible Abstract Syntax Tree (AST) from the text files and write the tree representation back to a file. Currently we have no icon based DSL, which is best viewed using an icon-based editor. However, we implemented a connection editor allowing us to connect wind turbine components represented in the purely parametric representation like it is also done in many tools for Modelica models [4, 7].

³<http://www.apdl.de/>

⁴<http://www.mscsoftware.com/products/cae-tools/msc-nastran.aspx>

In the next section the transformation between different kinds of models is discussed focusing on the generation of Modelica code from wind turbine models.

3 Model Transformation

Since model driven software development is increasingly accepted and used by software engineers, transformations of the developed models are becoming important. Various techniques for the transformations have been developed, of which some are described in this section. The transformations to Modelica models which are used in our project are presented in the subsequent section.

3.1 About Model Transformations

Model transformations can be done in different ways. The most appropriate one is the direct transformation between models based on rules defined for elements of two meta-models. These rules can automatically be applied to convert one model to another. If the transformation rules are bijective, i.e., in both directions, automatic synchronization between two models can be realized. This kind of transformation is called a Triple Graph Grammar (TGG) [10].

TGGs can only be defined for a small set of models. The first requirement is, that the two models to be transformed into each other must be semantically similar. For example, models of towers can not be transformed to rotor blade models. Secondly the information content must be comparable. Modal blade models may not be translated into more detailed models that can be used for FEM simulations. The information needed for the physical properties in a FEM model cannot be automatically derived from the kind of parameters available in a modal blade model. This observation does not only hold in the context of TGGs. Generally speaking, a transformation from one model to another can only be done if the “structural information” content of the initial model is greater or equal to the “structural information” of the target model.

In general models of physical components at different levels of detail for the use with different theories do not meet the above mentioned requirement for TGGs of being bijective. Often physical theories are needed to transform detailed models into coarser models and the opposite design transformation is always based on assumptions and engineering know how, which is to be obtained from the user in terms of parameters of

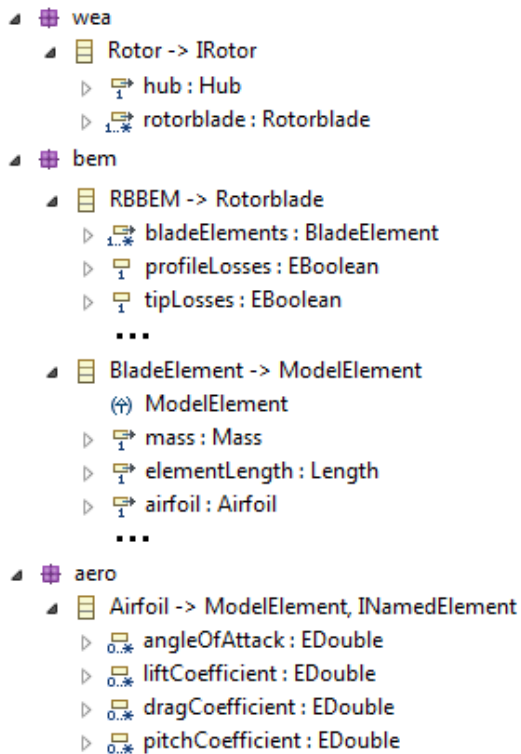


Figure 3: EDD-model of a Rotor, Rotor Blade, Blade Element and Airfoil

the transformation. Transformation mechanisms are needed to implement the complex algorithms in order to perform the transformations. Nevertheless the transformation of a detailed model into a coarse model is highly automatable and can be reused when parameters change in the detailed model during the design process. For the transformation of EMF based models several tools are available, like QVT or ATL⁵. These languages provide functional language style syntax for the definition of automatic model transformation rules. In the future these languages might be used in the OneWind project where applicable. However, at the moment only Java-based transformation modules are being developed.

3.2 Transforming EDD to Modelica Code

As a result of defining the Modelica language as a Xtext grammar, serialization/unparsing of Modelica code from an AST representation to textual Modelica source code is automatically available. Transforming EMF based wind turbine models to Modelica AST representation may be possible by using transformation languages as mentioned above. However, since the abstract syntax of Modelica is rather complex, our initial

approach is to write such transformers in Java.

The generated Modelica source files are used along with the OneWind library (see Figure 4) mentioned in section 1 for the highly coupled aero-servo-hydro-elastic simulation of wind turbines. The EDD model for a rotor (see Figure 3) is explained and serves as an example of the generation of Modelica code. A rotor consists of a hub and multiple rotor blades. Usually three rotor blades are used in modern horizontal axis wind turbines. The rotor blades consist of blade elements that define structural properties like masses, stiffnesses or lengths. Additionally each blade element defines an airfoil that describes the aerodynamic properties of that part of the blade. The user can edit the properties mentioned above, in order to design a rotor. The generator then generates Modelica records containing the user defined parameters. A Modelica rotor stub that is defined in the OneWind library is parameterized by the generated data. Finally the model consisting of the library components and the generated part can be simulated using Modelica simulation environments. The parameters which are customizable by the user are separated in Modelica records. Hence for each model that is being transformed, e.g. a blade, a data record is created that contains the parameters. In the blade example the data record contains single parameters for unary properties and arrays for multiple properties like blade elements. The array size is equal to the number of elements in the list. Moreover, the user can choose between different kinds of components to change the structural properties of the model. One can, for example, decide whether a rigid, modal

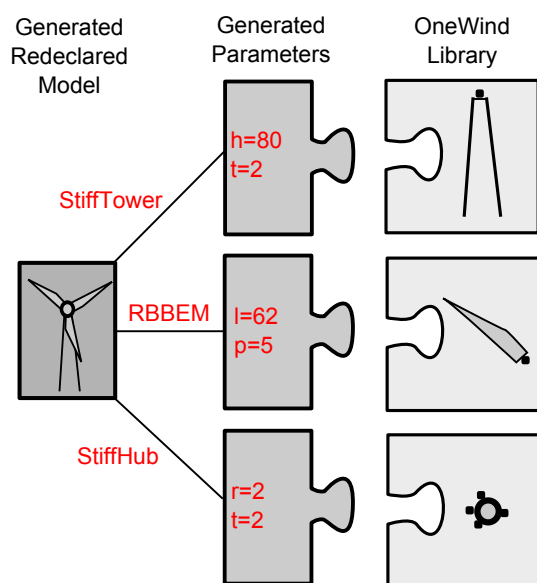


Figure 4: The OneWind Library and Generated Modelica Code

⁵<http://www.eclipse.org/m2m/>

or FEM blade model shall be used for the simulation. When the user selects a blade model that differs from the default one used in the library, the blade model is changed by the generation of a redeclaration. Thereby it is possible to customize the wind turbine model. The approach described above allows us to provide the wind turbine designer with an abstract view of the main properties of a wind turbine model. Variants of wind turbine models can be created quickly and compared to each other.

In the next section some problems that occurred during the implementation of the transformation modules including the use of polymorphic data types are described.

4 Polymorphism in Modelica

The transformation strategy described in the previous section provides a generic way of converting models from a purely parametric representation to simulatable Modelica models. For this approach to work the two types of models must be structurally equivalent. Hence the library must be designed in a way that provides suitable class stubs for the generation of Modelica code and the parametric model must meet the structure of the components defined by the library. During the development, we recognized that the conditions can be met with reasonable effort. Nevertheless problems arise in cases where lists of items of complex types are transformed to Modelica code. One example is rotor blades containing blade elements that have a length and an airfoil property. In the example, the *NREL 5 MW reference baseline wind turbine model* [9] is used. Listing 1 displays the resulting data record of the transformation. A load element contains a parameter profile that holds a list of aerodynamic

Listing 1: Generated Blade Data Record

```
record BladeData
  //array length 3
  Profile_Cylinder1 cylinder1;
  //array length 142
  Profile_DU21 du21;
  //array length 127
  Profile_NACA64 naca64;
  parameter Integer nBladeElements = 3;

  replaceable parameter Profile
  profile[5] =
  {cylinder1, du40, du35, du25, naca64};

end BladeData;
```

profiles (Listing 2) of type Profile. The class Profile displayed in Listing 3 is a kind of template record. It contains arrays of profile specific data: the angle of attack $\alpha[\text{deg}]$, lift coefficient $ca(\alpha)[-]$, drag coefficient $cw(\alpha)[-]$ and pitching moment coefficient $cm(\alpha)[-]$. The array size is variable as the number of properties varies between different profile types.

Listing 2: Load Element Containing the Airfoil Descriptions

```
model LoadElement
  parameter Profile profile;
end LoadElement;
```

Concrete profile records like Profile_NACA64 (see Listing 4) define the profile specific value quantity and assign the concrete values to the array. This structure provides a similar behaviour as generic array lists in Java whereby the generic type in this case is defined by the Modelica record Profile.

Listing 3: Generic Type Profile

```
record Profile

  import Modelica.SIunits.Conversions.
  NonSIunits.Angle_deg;
  parameter Angle_deg alpha[:];
  parameter Real caOfAlpha[:];
  parameter Real cwOfAlpha[:];
  parameter Real cmOfAlpha[:];

end Profile;
```

For simplification reasons the listing shows only a reduced set of aerodynamical coefficients of the NACA64 airfoil. Typically, it consists of many support positions for the complete range (-180 to +180 deg) of the attack angle α with variable equidistant steps. For frequently used regions of attack angles usually small step sizes are used. Thereby a better linearisation between these points and approximation of the measured coefficients during a simulation can be achieved for the used airfoil. The profile data can be used in a unified way as defined by the Profile record and therefore the class using the profile data does not need to know the concrete profile type. Finally a blade model is generated that assigns the aerodynamic profiles to the load element of the blade (Listing 5). A problem that occurs using polymorphic arrays as explained above is that the created list of instances of class Profile consists of types with different array size, e.g. the size of all

arrays from `profile[1] = 3` and from `profile[2] = 142`. Unifying the records leads to ragged arrays that are not defined by the Modelica specification. Hence the behaviour during simulation is unpredictable or the simulation tool does not even compile the code.

Listing 4: Concrete Profile Record Profile_NACA64

```
//For clarity reduced profile set
//of NACA64 (15 instead of 127 values)
record Profile_NACA64
  extends Profile
  (
    alpha = {-180.00,-90.00,-30.00,
             -10.00,-5.00,-3.00,-1.00,
             0,1.00,3.00,5.00,10.00,
             30.00,90.00,180.00},
    ca0fAlpha = {0,-0.067,-0.829,
                 -0.711,-0.151,0.088,
                 0.328,0.442,0.556,
                 0.784,1.011,1.382,
                 0.926,0.053,0},
    cw0fAlpha = {0.0198,1.3587,0.4295,
                 0.0111,0.0079,0.0064,
                 0.0052,0.0052,0.0052,
                 0.0053,0.0058,0.015,
                 0.4294,1.4565,0.0198},
    cm0fAlpha = {0.00,0.3636,0.1563,
                 -0.0734,-0.0841,-0.0912,
                 -0.0971,-0.1014,-0.1076,
                 -0.1157,-0.124,-0.1149,
                 -0.1668,-0.3858,0.00};
  );
end Profile_NACA64;
```

Listing 6 shows a workaround for this issue. Instead of creating objects from a list of types with variable array size, the Profile class for each load element object is directly declared with a modification statement of the desired profile class. The profile data of each blade element object is filled by array concatenation which corresponds to a normal parameter modification statement. This circumvents getting objects with variable sized array types. In this case a list of instances of the class LoadElement is defined, where each load element object has a different airfoil type and the size is specified by the respective modification. Thus the array sizes are known at this point as the profile data is not assigned in a generic way and the compiler does not fail to unify the record types.

The drawback of this approach is that the code generation is not realized as described in section 3.2 and therefore it may not be obvious where the generated data comes from. Additionally it prevents one from creating an automatic transformation mechanism as custom adaptations to the code generators must be implemented. The advantage of an automatic

transformation algorithm is that it reduces the implementation effort, creates code that is easier to test and it enhances the readability of the generated code.

Listing 5: Generated Blade Model

```
model NREL5MBlade

  BladeData bladeData;
  LoadElement loadElement
    [bladeData.nBladeElements](
      profile = bladeData.profile
    );

end NREL5MBlade;
```

The model from Listing 6 can now replace the default blade model from the OneWind library by using Modelica replaceable object types. This also holds true for the class LoadElement. The physical algorithms (e.g. calculating loads for the blade from wind inflow) are reused, only the calculation parameters are modified. This approach is used for all main components of the library (rotor, nacelle, tower, substructure, operating control, environment etc.) in order to create a custom model of a wind turbine.

Listing 6: Redeclaration of Blade Element Data

```
model NREL5MBlade
  extends RigidBlade(
    redeclare LoadElement loadElement(
      // old:
      // profile = bladeData.profile
      // new:
      profile = {
        bladeData.cylinder1,
        bladeData.cylinder1,
        bladeData.cylinder2,
        bladeData.du40,
        bladeData.du35,
        // ...
        bladeData.naca64,
        bladeData.naca64}
      )
    );
end NREL5MBlade;
```

The OneWind library contains a default wind turbine model `HorizontalAxis.OffshoreWindTurbine`. All main components in this model are replaceable objects and can thereby be redeclared by parameterised classes of the concrete NREL5M model. Listing 7 shows the main class of the generated concrete wind turbine model of the *NREL5M reference baseline offshore wind turbine*, which inherits from the default model. It can be simulated with a Mod-

elica compiler in combination with the generated model classes and the OneWind library components.

Listing 7: Main wind turbine class with redeclared components

```

model NREL5MOffshore
  extends
    HorizontalAxis.OffshoreWindTurbine
  (
    redeclare NREL5MoperatingControl
      operatingControl,
    redeclare NREL5MRotor rotor,
    redeclare NREL5Mnacelle nacelle,
    redeclare NREL5MTower tower,
    redeclare NREL5MSubstructure
      substructure,
    redeclare NREL5MSoil soil,
    redeclare NREL5MWind wind,
    redeclare NREL5MWater water
  );
end NREL5MOffshore;

```

5 Conclusion and Outlook

Based on the experience gained during the development of our simulation environment, we can see that it is possible to create a common data basis for different tools dealing with the design and simulation of wind turbines. Transformations between different kinds of models enable the re-use for different purposes.

As described for the generation of Modelica models, the use of the EDD approach allows one to parameterize models and to create simulatable representations like Modelica source code. Furthermore, models can be structurally customized to create several versions of physical models in the end. In the future, automatic transformation with languages as described in section 3 may be introduced for the transformation between EDD models as well as between EDD and simulator specific models.

For the Modelica code generation it is desirable to use transformation languages, since changes in the Modelica language specification are easier reflected by adapting a few transformation rules than by modifying Java classes.

Increased polymorphism, as discussed in section 4, would enhance the generation as the generated code would be easier to understand and the generation could be encapsulated for each component. This simplifies the code generation and reduces the dependencies between components and their contained declarations.

In order to enable the use of more polymorphism, the Modelica language specification would have to be

enhanced, in this case allow polymorphic ragged arrays. As most of the Modelica simulators compile Modelica to plain C code, the polymorphism would have to be adapted as C does not support polymorphism. However, this could enhance the parameterization of Modelica code as the implementation would be independent from the concrete components that are used.

To enhance the transformation process we will investigate the use of transformation languages as the next step. This will provide a more generic way of code generation and enhance the maintainability since changes in the meta-model of Modelica can be applied easier.

Furthermore, investigation is needed whether transformation rules can be derived that allow transformation of arbitrary types of models. Hence, it would not be necessary to create transformation rules for each particular EDD model, but universally applicable rules would further simplify the transformations. To realize this goal more generic data structures as described in section 4 would be desirable.

Acknowledgment

This work is financially supported by the Federal Ministry for the Environment, Nature Conservation and Nuclear Safety based on a decision of the Parliament of the Federal Republic of Germany.

References

- [1] Frank Budinsky, Stephen A. Brodsky, and Ed Merks. *Eclipse Modeling Framework*. Pearson Education, 2003.
- [2] Eric Clayberg and Dan Rubel. *Eclipse Plug-ins*. Addison Wesley Professional, 3rd. edition, 2009.
- [3] International Electrotechnical Commission. *Wind turbines - part 3: Design requirements for offshore wind turbines*, 2009.
- [4] Dynasim AB. *Dymola User Manual: Version 6*. Dassault Systemes, Lund (Sweden), 2009.
- [5] Germanischer Lloyd Wind Energie. *Guideline for the certification of offshore wind turbines*, 2005.
- [6] Martin Fowler. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*.

- Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 3 edition, 2003.
- [7] Peter Fritzson, Adrian Pop, Martin Sjölund, Per Östlund, and et. al. Openmodelica useres guide: Version 2012-01-30 for openmodelica 1.8.1. Linköping (Sweden), January 2012.
- [8] Richard C. Gronback. *Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit*. Addison-Wesley, 2009.
- [9] J Jonkman, S Butterfield, W Musial, and G Scott. Definition of a 5-mw reference wind turbine for offshore system development. *Contract*, 324(February):75, 2009.
- [10] Alexander Königs. Model transformation with triple graph grammars. In *Model Transformations in Practice, Satellite Workshop of Models 2005, Montego*, 2005.
- [11] Donald E. Knuth. Top-down syntax analysis. *Acta Informatica*, 1:79–110, 1971.
- [12] Jan Köhnlein and Sven Efftinge. Xtext 2.1 documentation, October 31, 2011.
- [13] Jeff McAffer, Paul VanderLei, and Simon Archer. *OSGi and Equinox: Creating Highly Modular Java Systems*. Addison-Wesley Professional, 1st edition, 2010.
- [14] Modelica Association. The Modelica Language Specification version 3.2, 2010.
- [15] OMG. *Meta Object Facility (MOF) Core Specification Version 2.0*, 2006.
- [16] Roland Samlaus, Claudio Hillmann, Birgit Demuth, and Martin Krebs. Towards a model driven modelica ide. In *8th International Modelica Conference*, 2011.
- [17] M. Strach, F. Vorpahl, C. Hillmann, M. Strobel, and M. Brommundt. Modeling offshore wind turbine substructures using engineer design data — a newly developed approach. In *Proceedings of the Twenty-second (2012) International Offshore and Polar Engineering Conference*, Rhodes, June 2012. International Society of Offshore and Polar Engineers (ISOPE). Accepted.
- [18] M. Strobel, R. Vorpahl, C. Hillmann, X. Gu, A. Zuga, and U Wihlfahrt. The onwind modelica library for offshore wind turbines - implementation and first results. In *Proceedings of the Modelica Conference*, 2011.

Derivative-free Parameter Optimization of Functional Mock-up Units

Sofia Gedda^{a,c} Christian Andersson^{a,c} Johan Åkesson^{b,c} Stefan Diehl^a

^aCentre for Mathematical Sciences, Lund University, Sweden

^bDepartment of Automatic Control, Lund University, Sweden

^cModelon AB, Sweden

Abstract

Representing a physical system with a mathematical model requires knowledge not only about the physical laws governing the dynamics but also about the parameter values of the system. The parameters can sometimes be measured or calculated, but some of them are often difficult or impossible to obtain directly. Nevertheless, finding accurate parameter values is crucial for the accuracy of the mathematical model.

Estimating the parameters using optimization algorithms which attempt to minimize the error between the response from the mathematical model and the real physical system is a common approach for improving the accuracy of the model.

Optimization algorithms usually require information about the derivatives which may not always be easily available or which may be difficult to compute due to, e.g., hybrid dynamics. In such cases, derivative-free optimization algorithms offer an alternative for design and parameter optimization.

In this paper, we present an implementation of derivative-free optimization algorithms for parameter estimation in the JModelica.org platform. The implementation allows the underlying dynamic system to be represented as a *Functional Mock-up Unit (FMU)*, and thus enables parameter optimization of models exported from modeling tools compliant with the *Functional Mock-up Interface (FMI)*.

Keywords: Derivative-free optimization; Parameter Estimation; JModelica.org; FMI; Assimulo

1 Introduction

Increasingly, industry rely on mathematical modeling for evaluating and designing new machines and devices. As the models grow increasingly complex, the

need for estimating parameters which are unknown or uncertain is put into focus. Estimating unknown parameters in the mathematical model using optimization algorithms is a commonly used approach to increase the accuracy of models. In this paper, we focus on parameter estimation problems where the objective is to minimize the error between the simulated profiles of the mathematical model and measurements from the corresponding physical system. The objective function considered

$$f(x) = \sum_{i=0}^M (y^{\text{sim}}(t_i, x) - y^{\text{meas}}(t_i))^2 \quad (1)$$

where y^{sim} is the model output trajectory and y^{meas} are the measurements. The parameters to be estimated are $x \in \mathbb{R}^n$, where n is the number of parameters. M is the number of measurements at the time points t_i . The optimization problem is then formulated as

$$\min_{x \in \mathbb{R}^n} f(x). \quad (2)$$

subject to the system dynamics, in the FMI case given by a hybrid Ordinary Differential Equation (ODE). Additionally, the parameters may be subject to bounds, $l \leq x \leq u$.

This optimization problem may be solved by transcribing the problem into a non-linear programming problem using either shooting methods [6] or collocation methods [6]. These methods, however, both use derivative information, which may be difficult or expensive to compute, e.g., in the case of hybrid systems. The idea is then to use algorithms which do not depend on derivative information, such as the Nelder-Mead simplex method [7]. In a derivative-free method, instead of using information from the derivatives to improve the solution, the objective is evaluated at a chosen set of points which are then used to improve

the solution. How the points are chosen and which strategy is used to improve the solution depends on the method. Typically, computation times are longer than for derivative-based methods, but on the other hand, derivative-free methods offer a feasible and robust option when other algorithms fail.

In this paper, we evaluate three derivative-free optimization algorithms for parameter estimation available in the JModelica.org platform: the Nelder-Mead simplex method, the differential evolution method and a genetic algorithm. Based on this evaluation, the Nelder-Mead algorithm seems most appropriate to solve the class of parameter optimization problems considered.

The main contribution of the paper is an implementation of the Nelder-Mead simplex algorithm. The algorithm supports parameter bounds and parallel evaluation of function evaluations where FMU models are loaded and simulated.

We also briefly present the underlying packages FMI Library (FMIL), PyFMI¹ and ASSIMULO². These packages are part of JModelica.org, but also available stand-alone, and are used for simulating the model response. In Figure 1, an overview of the interaction between the packages in JModelica.org when solving a derivative-free optimization problem is shown.

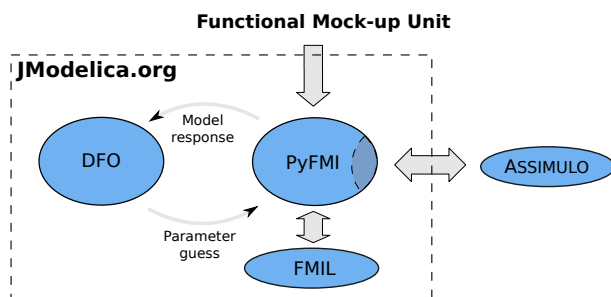


Figure 1: Overview of the interaction between the packages in JModelica.org when solving a derivative-free optimization problem.

The paper is outlined as follows. In Section 2, the *Functional Mock-up Interface* is presented together with an overview of optimization tools. In Section 3, an introduction to the JModelica.org platform is given together with the simulation package ASSIMULO as well as the Python package PyFMI for interaction with FMUs. Next, derivative-free optimization algorithms are introduced, followed by a description of the implementation in JModelica.org. In Section 6, the imple-

¹<http://www.pyfmi.org>

²<http://www.assimulo.org>

mentation is applied to two different problems where the second is a large industrial example where a model of an engine is calibrated. Finally, Section 7 concludes the paper with a summary and conclusions.

2 Background

2.1 The Functional Mock-up Interface

The *Functional Mock-up Interface* [1] defines an open standard for model exchange. The intention is to allow exchange of models between different modeling and simulation tools. The standard describes models as hybrid ODEs with state, step and time events. A model that implements the FMI standard is called a *Functional Mock-up Unit* and is distributed as a compressed directory containing a shared object file or source code containing the model equations, and a set of functions for data access, and an XML file, which describes the model parameters and variables. The standard has received a significant amount of attention among vendors since the release in 2010 and currently there are 34 environments that support or plan to support the standard.

2.2 Optimization tools

There exist many tools for optimization of complex systems, both in the public domain and commercially available. Broadly, there are three different categories of optimization tools, although the scope is sometimes overlapping. In *Model integration tools* the problem of interfacing several design tools into a single computation environment, where analysis, simulation and optimization can be performed is addressed. Examples include ModelCenter [23], OptiY [22], modeFRONTIER [12], and iSIGHT [10]. Such environments are capable of integrating several simulation and design tools into one computational chain, where the results are optimized. The integrated tools may be heterogeneous in the sense that they model different physical domains by means of different algorithms. Due to this heterogeneity amongst supported tools, optimization algorithm that does not exploit derivatives or model structure such as sparsity is commonly employed. Model integration tools typically also have strong support for model approximation and visualization.

Many *modeling and simulation tools* has optimization add-ons, e.g., Dymola [9], gPROMS [24], Jacobian [19], and OMOptim [18]. The level of support for optimization in this category differs between the

tools. Dymola, for example, offers add-ons for parameter identification and design optimization [11, 20]. gPROMS on the other hand, offers support for solution of optimal control problems and has the additional benefit in comparison with Modelica tools to provide support for partial differential equations (PDEs). Tools in this category usually support a set of derivative-based and derivative-free optimization algorithms. Optimization problems are typically formulated by means of graphical user interfaces.

In the third category there are *numerical packages* for dynamic optimization, often developed as part of research programs. Examples are ACADO [21], Muscod II [28], and DynoPC [17]. Such packages are typically focused on efficient implementation of an optimization algorithm for a particular class of dynamic systems. Also, detailed information about the model to optimize is generally required in order for such algorithms to work, including accurate derivatives and in some cases also sparsity patterns. While these packages offer state of the art algorithms, they typically come with simple or no user interface. Their usage is therefore limited due to the effort required to code the model and optimization descriptions. A notable example is CasADi [4], which provides an efficient AD kernel, interfaces to numerical optimization algorithms and a comprehensible Python interface for custom development of dynamic optimization algorithms. CasADi also support import of Modelica models in XML format, see [5].

The approach presented in this paper falls into the category of additions to modeling and simulation tools. Specifically, models exported from FMI compliant tools can be optimized. The presented algorithm uses Python scripting as a means to formulate optimization problems, and in this respect it differs from, e.g., the approach taken in Dymola.

3 JModelica.org

JModelica.org³ [26] is a platform for modeling, simulation and optimization of complex physical systems primarily based on the Modelica⁴ modeling language. JModelica.org is a community-based open-source project started at Lund University with the following aim:

“To offer a community-based, free, open-source, accessible, user and applica-

³<http://www.jmodelica.org>

⁴<http://www.modelica.org>

tion oriented Modelica environment for optimization and simulation of complex dynamic systems, built on well-recognized technology and supporting major platforms.”

JModelica.org provides compilers for the Modelica language and the extension Optimica [25]. For simulations, the Python package ASSIMULO is used for both simulating ODEs and DAEs. Dynamic optimization is available using direct local collocation algorithms based on the DAE formulation of the model. The user interaction with JModelica.org is based on the programming language Python.

Included in JModelica.org are packages that can also be used stand-alone. In the following subsections, the packages FMI Library, PyFMI and ASSIMULO are presented.

3.1 FMI Library

FMI Library (FMIL) is a C package designed for working with FMUs and serving as support for applications interfacing the FMI. The package contains convenient methods for decompressing of FMUs, parsing XML information and connecting the binary⁵. The library supports FMI 1.0 for model exchange and for co-simulations and is intended for custom integration of FMI technology in applications. FMIL is also used as a basis of the Python package PyFMI.

3.2 PyFMI

PyFMI [2] is a package for interacting with FMUs using Python, based on the FMI Library. It provides convenient high-level functions for interacting with an FMU, retrieving values and accessing variable information from the XML information. Additionally, a low-level mapping of the functions specified in the interface can also be accessed. A model can be loaded and made available from Python using the following Python code:

```
#Import the model class
from pyfmi import FMUModel

#Load the model into Python
model = FMUModel("bouncingBall.fmu")
```

PyFMI also provides a connection to the simulation package ASSIMULO and thus enables access to state-of-the-art solvers such as CVode and IDA from the Sundials suite, capable of simulating hybrid systems.

⁵<http://www.jmodelica.org/FMILibrary>

A simulation is performed by using the `simulate` method.

```
#Simulate the model using Assimulo
res = model.simulate(final_time=10)
```

3.3 ASSIMULO

ASSIMULO [3] is a Python package for solving first or second order explicit ordinary differential equations (ODEs) or implicit ordinary differential equations (DAEs).

ASSIMULO combines a variety of different solvers written in FORTRAN, C and Python via a common high-level interface. The state-of-the-art solvers CVode and IDA from the SUNDIALS suite [15] as well as RADAU5 [14] are amongst the available solvers.

ASSIMULO is divided into two parts, namely problem definitions and solvers. A problem definition may in addition to the right-hand side of the differential equation also contain for instance the Jacobian as well as event functions in order to support simulation of hybrid systems. The idea is to separate information related to a problem from the solver. For instance, which states are algebraic is information that is related to the problem and not the solver. In Figure 2, an overview is given showing the available problem definitions and solvers in ASSIMULO. Also shown is the connection between the different problem formulations.

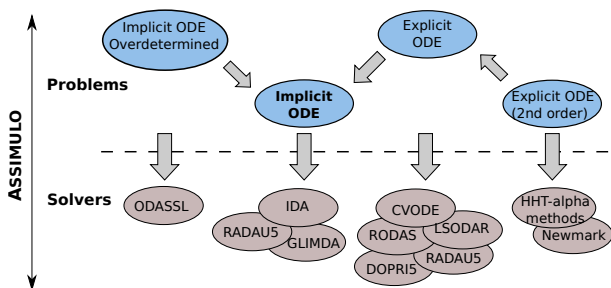


Figure 2: Connection between the different problem formulations and the different solvers available in ASSIMULO.

4 Derivative-free Optimization

In applications where derivatives are difficult or computationally expensive to obtain, there is a need for derivative-free optimization methods. Examples include very large models which also contains hybrid elements.

δ	Operation type
$-\frac{1}{2}$	inside contraction
$\frac{1}{2}$	outside contraction
1	reflection
2	expansion

Table 1: Different δ -values with corresponding operation types.

We shall now introduce three different derivative-free optimization algorithms which have been implemented or interfaced in the JModelica.org platform [13]: the Nelder-Mead simplex method, the differential evolution method and a genetic algorithm.

4.1 The Nelder-Mead simplex method

The Nelder-Mead simplex method has obtained its name from the fact that each iteration is based on a simplex. A simplex in \mathbb{R}^n is a set of $n+1$ vertices $x_1, \dots, x_{n+1} \in \mathbb{R}^n$ such that the vectors $x_i - x_1, i = 2, \dots, n+1$ are linearly independent, i.e. it is a generalization of a triangle to arbitrary dimension.

In each iteration of the Nelder-Mead algorithm, the objective is to replace the vertex with the highest cost in the n -dimensional simplex with a better point. The vertices are ordered by increasing value of f such that $f(x_1) \leq \dots \leq f(x_{n+1})$. The new point is searched for along the line through the vertex with the highest cost, x_{n+1} , and the centroid,

$$x_c = \frac{1}{n} \sum_{i=1}^n x_i, \quad (3)$$

of the remaining vertices x_1, \dots, x_n . This line has the equation

$$x = x_c + \delta(x_c - x_{n+1}), \quad \delta \in \mathbb{R}. \quad (4)$$

The parameter δ defines the type of the operation performed on the simplex. There are four different operation types that are performed by the algorithm: reflection, expansion, inside contraction or outside contraction, resulting in the reflection point, x_r , the expansion point, x_e , the inside contraction point, x_{ic} , or the outside contraction point, x_{oc} respectively. Table 4.1 displays the δ -values corresponding to these four operations. If none of these operations results in a better point than x_{n+1} , the simplex is shrunk toward the vertex with the lowest cost, x_1 . That is, the n points with the highest costs are replaced by new points obtained

from

$$x = x_1 + \frac{1}{2}(x_i - x_1), \quad i = 2, \dots, n + 1. \quad (5)$$

This procedure is repeated until some termination criterion is fulfilled. There are usually three different termination criteria, one of which has to be fulfilled in order for the algorithm to terminate:

- Convergence criterion for x – the simplex is sufficiently small according to a user-provided tolerance.
- Convergence criterion for f – the function values at the simplex vertices are sufficiently close according to a user-provided tolerance.
- Termination criterion without convergence – the maximum number of iterations or function evaluations has been reached.

In Figure 3, two iterations of the algorithm are shown, illustrating how the simplex changes form and position.

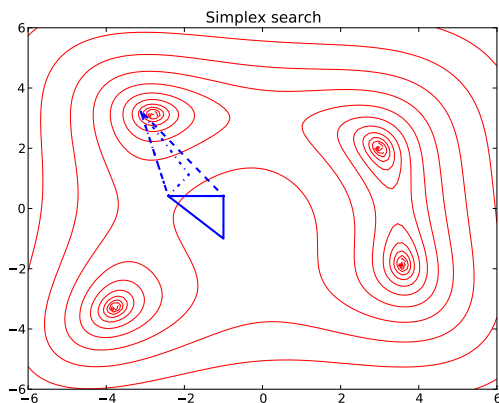


Figure 3: Two simplex iterations where the solid triangle is the initial simplex which transforms into the dashed triangle ($\delta = 2$) and then the dash-dot triangle ($\delta = -\frac{1}{2}$).

4.2 Evolutionary algorithms

The differential evolution method and genetic algorithms belong to the class of evolutionary algorithms, which consists of stochastic optimization algorithms inspired by the principles of biological evolution theory. In such algorithms, each candidate solution, $\bar{x} \in \mathbb{R}^n$, represents an individual and the objective function, $f(x)$, or fitness function, represents the environment

within which the individuals live. The value $f(\bar{x})$ determines how fit the individual \bar{x} is to survive in the environment; a lower value means a better fit. At each iteration, or generation, a new population of possible solutions is produced through mutation, crossover and selection. Mutation is a mechanism for maintaining genetic diversity by modifying an existing solution while crossover means combining two existing solutions into a new one.

4.2.1 The differential evolution method

The differential evolution method [27] works according to the following steps.

Initialization: An initial population of N individuals, or vectors, is generated randomly inside the feasible region.

Mutation: At each iteration, the population consists of N vectors, $x_i \in \mathbb{R}^n, i = 1, \dots, N$. For each vector x_i , the target vector, a mutant vector, $v_i \in \mathbb{R}^n$, is produced by adding the weighted difference between two vectors in the current population to a third one according to the following formula:

$$v_i = x_{r_1} + F(x_{r_2} - x_{r_3}),$$

where $r_1, r_2, r_3 \in \{1, 2, \dots, N\}$ are random indices, distinct from each other and from i , and $F \in [0, 2]$ is a constant.

Crossover: The mutant vector, v_i , is then recombined with its corresponding target vector, x_i , through a mixing of their elements, generating a trial vector, $u_i \in \mathbb{R}^n$. The trial vector receives elements of the mutant vector with probability $P \in [0, 1]$ and elements of the target vector with probability $1 - P$.

Selection: The trial vector, u_i , is compared with the target vector, x_i , and the one giving the lowest value of the fitness function, f , is selected for the next generation.

The phases mutation, crossover and selection continue until a termination criterion is fulfilled.

4.2.2 Genetic algorithms

In genetic algorithms [16] the individuals are encoded as bit strings. There are various genetic algorithms which differ from one another but the following is a general description.

Initialization: An initial population of size N is generated randomly inside the feasible region.

Selection: In each generation, a selection probability, $p(x_i)$, is defined for each individual, $x_i \in \mathbb{R}^n$. The selection probability depends on the fitness function

value for the individual, $f(x_i)$, a smaller value gives a larger probability. Two individuals are then selected randomly according to their selection probabilities.

Crossover: Crossover is performed on the two selected individuals with a certain probability, the crossover rate. A common choice for this probability is around 0.7. There are different crossover techniques but a common approach is to randomly choose a position in the bit strings and swap all bits between the two strings after that position.

Mutation: Mutation is performed by flipping bits (from 0 to 1 or vice versa) at random positions in the bit strings. The probability of flipping a bit, the mutation rate, should be much lower than the crossover rate.

Selection, crossover and mutation is repeated until a termination criterion is reached.

5 Implementation

The algorithms evaluated in Section 4, have been made available in JModelica.org. The Nelder-Mead simplex algorithm has been implemented and is now provided as part of JModelica.org, while the differential evolution algorithm and a genetic algorithm has been interfaced through the OpenOpt package⁶. The algorithms are available through the Python function `fmin` in JModelica.org.

The method `fmin` requires as input the objective function together with the initial conditions as well as options for specifying the intended optimization algorithm and tolerances. In Section 6.1, it is shown how the objective function can be defined when the dynamic model is contained in an FMU.

In the Nelder-Mead algorithm, support for parallel evaluation of the objective function, $f(x)$, has been implemented. In each iteration of the algorithm, the evaluations of the $n + 1$ vertices are distributed over a user-supplied number of processes, as well as the evaluations of the reflection, expansion and contraction points.

For further implementation details, see [13].

6 Examples

In [13], the different derivative-free algorithms was tested and the result indicated that the Nelder-Mead algorithm is the preferred algorithm for the tested parameter estimation problems. The evaluation was done

⁶<http://openopt.org/>

based based on execution time and convergence to the optimal solution.

6.1 Furuta pendulum

The Furuta pendulum is a system consisting of a horizontal arm driven by a motor which is connected to a vertical pendulum, see Figure 4. The system has two degrees of freedom, namely the angle of the arm, ϕ , and the angle of the pendulum, θ . Additionally, there is friction in both the arm joint and the pendulum joint. Due to the discontinuities introduced by the friction, the system is not well suited for derivative-based optimization algorithms.



Figure 4: The Furuta pendulum.

The Furuta pendulum is modeled by a Modelica model, see Figure 5. The problem at hand is to calibrate the unknown friction coefficients of the arm and pendulum, respectively, against the given measurements using the Nelder-Mead simplex algorithm. The objective is thus

$$f(x) = \sum_{i=1}^M (\phi^{\text{sim}}(t_i, x) - \phi^{\text{meas}}(t_i))^2 + \sum_{i=1}^M (\theta^{\text{sim}}(t_i, x) - \theta^{\text{meas}}(t_i))^2 \quad (6)$$

where x is a vector containing the friction coefficients for the arm and the pendulum respectively.

The measurements were generated by simulation of the Modelica model for the Furuta pendulum and white measurement noise was added to the outputs. The measurements were given for a period of 40 seconds and were contained in a data file. The data was loaded into Python by the following code:

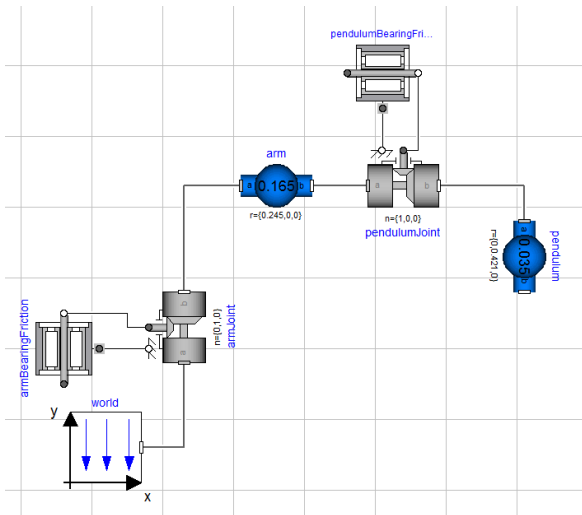


Figure 5: A Modelica model for the Furuta pendulum.

```

from scipy.io import loadmat
import numpy as N

# Load measurement data from file
data = loadmat('FurutaData')

# Extract data series
t_meas = data['time'][:,0]
phi_meas = data['phi'][:,0]
theta_meas = data['theta'][:,0]

y_meas = N.vstack((phi_meas, theta_meas))

```

The objective function is defined as a Python function where the FMU, generated by Dymola, for the Furuta pendulum is loaded and simulated for given parameter values.

```

from pyfmi import FMUModel
from pyjmi.optimization import dfo

# Define the objective function
def furuta_dfo_cost(x):
    #Scale down
    armFriction = x[0]/1e3
    pendFriction = x[1]/1e3

    # Load the FMU Model
    model = FMUModel('Furuta.fmu')

    # Set new parameter values
    model.set('armFriction',
             armFriction)
    model.set('pendulumFriction',
             pendFriction)

    # Simulate the model response
    res = model.simulate(final_time=40)

    # Load simulation result
    phi_sim = res['armJoint.phi']

```

```

theta_sim = res['pendulumJoint.phi']
t_sim = res['time']

# Evaluate the objective function
y_sim = N.vstack((phi_sim, theta_sim))
obj = dfo.quad_err(t_meas, y_meas,
                  t_sim, y_sim)

return obj

```

Finally, the objective is provided to the optimization function `fmin` together with the initial guess and the parameter bounds. The initial guess, i.e., the nominal values, were obtained through manual testing. The object returned by `fmin` contains the optimized parameters together with statistics, such as the number of iterations performed:

```

# Specify initial conditions (scaled)
x0 = N.array([0.012, 0.002]) * 1e3

# Lower and Upper bounds
lb = N.zeros(2)
ub = x0 + 10

# Solve using the Nelder-Mead algorithm
res = dfo.fmin(furuta_dfo_cost,
              xstart=x0, lb=lb, ub=ub,
              x_tol=1e-3, f_tol=1e-2)

# Optimal point rescaled
[armFriction_opt, pendFriction_opt] =
res[0]/1e3

```

The optimized parameter values were found to be 0.010 for the arm friction coefficient and 0.0010 for the pendulum friction coefficient. The result is visualized in Figure 6, where it can be seen that the model response is significantly more accurate using the optimized parameters as compared to the response given from the nominal parameters. In Figure 7, the error is shown between the measurements and the simulated response using both the nominal parameters and the optimized parameters.

6.2 Diesel Engine

In this example, parameters in a model of an exhaust gas pipe in a diesel engine is calibrated against measurements. The model was developed in Dymola using the Engine Dynamics Library and models a 13 liters Volvo truck engine [8]. The energy of the exhaust gas after the combustion is converted to torque, before releasing the gas to the purification process. In Figure 8, an overview of the model is shown. The energy is converted into torque by two turbines, shown as two trapezoids, where the first drives a compressor at the

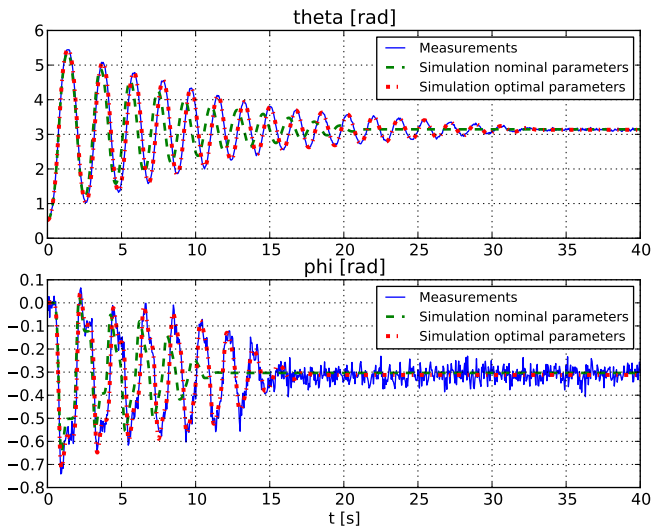


Figure 6: Simulation profiles corresponding to the optimized parameters (dashed-dotted), profiles resulting from simulation with nominal parameter values (dashed) and measurements (solid).

air intake of the engine and the second is connected to the drive shaft. Additionally, there are two gas volumes which are connected to two thermal conductors that transport heat to the surrounding air. The endpoint circles represent the boundary conditions for the gas.

The uncertain parameters are the thermal capacities in the walls of the gas volumes together with the thermal conductance from gas to wall in the volumes.

The inputs of the model are the gas temperature and pressure entering the system, angular velocity of the turbines and the gas pressure exiting the system. The output is the gas temperature exiting the system.

Measurements are provided for the inputs and the output sampled every second over a thirty minute period. In Figure 9, the result is shown when simulating the model using nominal parameter values.

The problem is to minimize the error between the simulated gas temperature that exits the system and the measured temperature,

$$\min_{x \in \mathbb{R}^n} \sum_{i=1}^M (T^{\text{sim}}(t_i, x) - T^{\text{meas}}(t_i))^2 \quad (7)$$

$$\text{subject to } x \geq 0 \quad (8)$$

where M is the number of measurement points and n the number of parameters.

Instead of optimizing the four uncertain parameters simultaneously, the problem is divided into two problems. The first problem is to determine the thermal capacity and the thermal conductance in the right volume. The second is to determine the thermal capacity

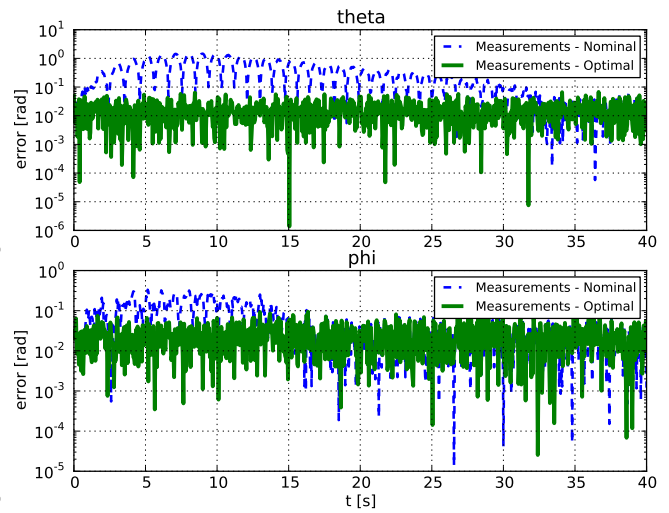


Figure 7: Error between the measurements and the simulated profiles using the nominal parameters (dashed) and the optimized parameters (dashed-dotted).

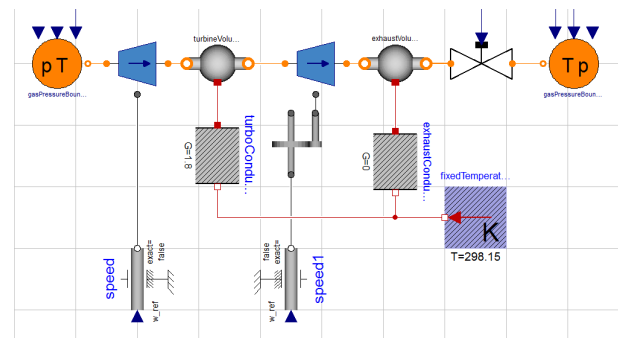


Figure 8: Overview of the model of the diesel engine.

and the thermal conductance in the left volume, using the results from the first problem. This procedure is used since the parameters of the first and second volume are correlated. Optimizing all parameters simultaneously then results in over-parameterization.

For each optimization problem, the first third of the measurement data sequences are used for calibration and the remaining part is used for validation.

The model was exported from Dymola as an FMU and thereby made available to the DFO algorithms in JModelica.org. The two problems are then solved using the Nelder-Mead simplex algorithm. Figure 9 shows the resulting simulation response for the optimized parameters. In Figure 10, the corresponding error profiles are shown for the calibration and validation data sets respectively. As can be seen, the optimized parameters significantly increase the accuracy of the model. The (scaled) RMS error was decreased from 1.0 to 0.18 for the calibration data set and from 1.0 to

0.36 for the validation data set.

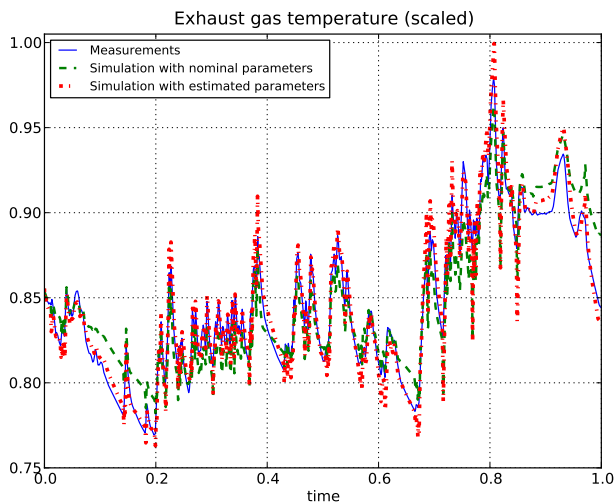


Figure 9: Simulation result with the optimized parameters together with result using the nominal parameter values and measurements.

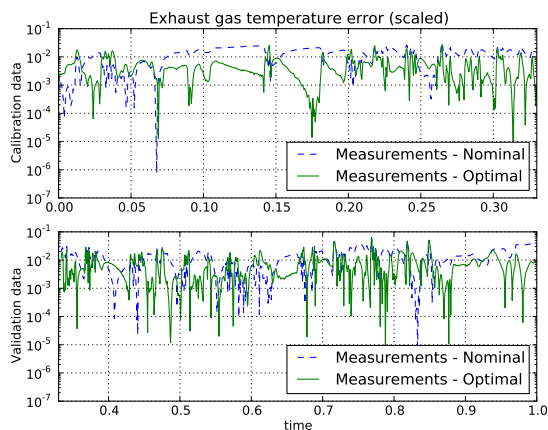


Figure 10: Error profiles for the calibration data set (top) and the validation data set (bottom).

7 Summary

An implementation of derivative-free optimization algorithms in JModelica.org has been presented. The implementation has been successfully applied to two dynamic models where the dynamics are contained in a *Functional Mock-up Unit*. In one of the examples, a Volvo truck engine was calibrated against measurement data, demonstrating the industrial applicability of the approach.

The Python-based user interface enables flexible implementation of complex cost functions involving,

e.g., simulation of FMUs and comparison with measurement data or algorithmic evaluation of complex discontinuous costs.

8 Acknowledgments

The authors gratefully acknowledges financial support from Vinnova under the contract (Project number P35278-5) and from the Lund Center for Control of Complex Systems, LCCC, funded by the Swedish Research Council.

References

- [1] Functional Mock-up Interface for Model Exchange. Interface specification, MODELISAR, January 2010.
- [2] C. Andersson, J. Åkesson, C. Führer, and M. Gäfvert. Import and export of Functional Mock-up Units in JModelica.org. In *8th International Modelica Conference 2011*. Modelica Association, 2011.
- [3] C. Andersson, J. Andreasson, C. Führer, and J. Åkesson. A workbench for multibody systems ODE and DAE solvers. In *The Second Joint International Conference on Multibody System Dynamics*, 2012.
- [4] J. Andersson, J. Åkesson, and M. Diehl. CasADi—A symbolic package for automatic differentiation and optimal control. In S. Forth, P. Hovland, E. Phipps, J. Utke, and A. Walther, editors, *Proc. 6th International Conference on Automatic Differentiation*, Lecture Notes in Computational Science and Engineering. Springer, 2012.
- [5] Joel Andersson, Johan Åkesson, Francesco Casella, and Moritz Diehl. Integration of casadi and jmodelica.org. In *8th International Modelica Conference*, March 2011.
- [6] T. Binder, L. Blank, H.G. Bock, R. Bulirsch, W. Dahmen, M. Diehl, T. Kronseder, W. Marquardt, J.P. Schlöder, and O. v. Stryk. *Online Optimization of Large Scale Systems*, chapter Introduction to model based optimization of chemical processes on moving horizons, pages 295–339. Springer-Verlag, Berlin Heidelberg, 2001.

- [7] A.R. Conn, K. Scheinberg, and L.N. Vicente. *Introduction to Derivative-Free Optimization*. Mps-siam Series on Optimization. Society for Industrial and Applied Mathematics/Mathematical Programming Society, 2009.
- [8] J. Dahl and D. Andersson. Gas exchange and exhaust condition modeling of a diesel engine using the Engine Dynamics Library. In *In 9th International Modelica Conference 2012*. Modelica Association, 2012.
- [9] Dassault Systèmes. Dymola Home Page, 2012. <http://www.3ds.com/products/catia/portfolio/dymola>.
- [10] Dassault Systèmes. iSIGHT Home Page, 2012. <http://www.3ds.com/products/simulia/portfolio/isight-simulia-execution-engine/overview/>.
- [11] H. Elmqvist, H. Olsson, S.E. Mattsson, D. Brück, C. Schweiger, D. Joos, and M. Otter. Optimization for design and parameter estimation. In *In 4th International Modelica Conference 2005*. Modelica Association, 2005.
- [12] ESTECO. modeFRONTIER Home Page, 2012. <http://www.esteco.com/>.
- [13] Sofia Gedda. Calibration of Modelica models using derivative-free optimization. Master’s thesis, Lund University, August 2011.
- [14] E. Hairer and G. Wanner. *Solving Ordinary Differential Equations: Stiff and differential-algebraic problems*. Springer series in computational mathematics. Springer-Verlag, 1993.
- [15] 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.
- [16] John H. Holland. *Adaptation in natural and artificial systems*. MIT Press, Cambridge, MA, USA, 1992.
- [17] Y.D. Lang and L.T. Biegler. A software environment for simultaneous dynamic optimization. *Computers and Chemical Engineering*, 31(8):931–942, 2007.
- [18] Linköping University. OMOptim Home Page, 2012. <https://openmodelica.org/index.php/developer/tools/176>.
- [19] Numerica Technology. Jacobian, 2012. <http://www.numericatech.com/jacobian.htm>.
- [20] H. Olsson, J. Eborn, S.E. Mattsson, and H. Elmqvist. Calibration of static models using Dymola. In *In 5th International Modelica Conference 2006*. Modelica Association, 2006.
- [21] OPTEC K.U. Leuven. ACADO Home Page, 2012. <http://www.acadotoolkit.org/>.
- [22] OptiY. OptiY Home Page, 2012. <http://www.optiy.de/>.
- [23] Phoenix Integration. ModelCenter Home Page, 2012. http://www.phoenix-int.com/software/phx_modelcenter.php.
- [24] Process Systems Enterprise. gPROMS Home Page, 2012. <http://www.psenderprise.com/gproms/index.html>.
- [25] Johan Åkesson. Optimica—an extension of modelica supporting dynamic optimization. In *In 6th International Modelica Conference 2008*. Modelica Association, March 2008.
- [26] 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. *Computers and Chemical Engineering*, 34(11):1737–1749, November 2010.
- [27] Rainer Storn and Kenneth Price. Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. *J. of Global Optimization*, 11(4):341–359, December 1997.
- [28] University of Heidelberg. MUSCOD-II Home Page, 2009. <http://www.iwr.uni-heidelberg.de/~agbock/RESEARCH/muscod.php>.

Stochastic Simulation and Inference using Modelica

Gregory Provan Alberto Venturini
Department of Computer Science,
University College Cork, Cork, Ireland
g.provan, a.venturini@cs.ucc.ie

Abstract

The physical modelling and simulation of systems with inherent uncertainty still poses significant issues when using Modelica and its tools. At present, both language and tools are fundamentally deterministic and offer limited support for handling uncertainty; this limits the scope of using Modelica in certain domains, e.g. feedback control systems. We propose a framework for incorporating uncertainty in Modelica simulation and analysis tasks. We do this by coupling a Modelica model with exogenous stochastic models. Finally, we apply this approach to the domain of building modelling.

Keywords: simulation; stochastic modeling; energy systems modeling

1 Introduction

Physical-model simulation using Modelica has traditionally been viewed as a deterministic problem, despite major sources of uncertainty. This uncertainty arises due to issues such as:

initial conditions incomplete input observations, measurement error, shortcomings in the data assimilation cycle, etc.

model accuracy and fidelity incomplete knowledge of physical processes (e.g., inaccurate parameterizations of sub grid-scale processes). incomplete and inaccurate numerical schemes,

At present, Modelica tools (e.g., Dymola) enable variability of initial conditions by different instantiations of model parameters Θ or by assigning values to internal model variables. However, this assignment can be done only once for each simulation. For simulations in which stochastic variables exist or there are external processes providing data (e.g., sensor/actuator data) to the model on a regular basis, the simulation must be re-started for each new input. This limits the

scope of using Modelica for use with certain feedback control systems (e.g., Model-Predictive control) or in embedded systems.

Consider the case in which Modelica currently deals with stochastic inputs, e.g., if we were to specify a probability distribution (pdf) over Θ . In this case, Monte Carlo (MC) sampling can be used to define a set of initial conditions for simulation. The drawback to this approach is that, for a complex pdf, a large number of samples (and hence simulations) will be needed in order to achieve a stochastically-sound set of simulations.

Throughout this article we will use the domain of energy modeling to explain our concepts. In particular, we will focus on the modeling of buildings, for which there exist several Modelica libraries, e.g., [13], for generating models for large, complex systems.

Our objective is to define a stochastic state evolution approach that is computationally efficient and can make use of existing Modelica deterministic simulators. We propose a framework for incorporating uncertainty in simulation and analysis tasks which use Modelica models. Our contributions are as follows:

- We propose a framework for ensemble-based stochastic optimisation, using Modelica as a deterministic modeling language and simulation methodology.
- We apply this approach to the domain of renewable energy in terms of underfloor heating optimisation.

Our approach shows how one can extend the existing Modelica language and toolset for such tasks. However, it also highlights deficiencies in Modelica for stochastic representation, as well as deficiencies in the Modelica tools to incorporate stochastic inference *within* a simulation, as well as the inability to accept exogenous inputs during a simulation.

2 Related Work

This work aims to extend both Modelica and Building Performance Simulation (BPS) with stochastic methods, and we discuss prior work in both areas.

Little work has focused on stochastic methods in Modelica. Most recently, Bouskela et al. [1] have described (a) methods for stochastic analysis and (b) proposals for identifying stochastic Modelica variables and performing appropriate inference. [11] discusses how a Modelica model can be used as a simulation model within computational design, such that the probability of a feasible design is explicitly computed.

In the area of BPS, Jacob et al. [9] integrate Monte Carlo sampling within embedded optimization for BPS. In particular, they use conditional probability density functions for energy consumption and demand to quantify the difference between a base case (of energy usage) with scenarios in the presence of uncertainty.

[8] shows how uncertainty analysis can improve BPS through a case study of an office building with respect to various building performance parameters, demonstrating the implications of uncertainty in results concerning energy consumption (annual heating and cooling) and thermal comfort (weighted over- and underheating hours).

One in-depth analysis of the impact of uncertainty in BPS, covering notions of *internal* and *external* probabilistic approaches to quantifying the overall effect of parameter uncertainty in building simulations, has been performed [12]. He quantifies the effects of uncertainty in building simulation by considering the internal temperature, annual energy consumption and peak loads. [3] study the potential impact of climate change on current building designs by examining future climates. They employ two methods, mathematical transformations of observed weather (morphing), and synthetic weather generator, to generate future weather files (on an hourly time scale) which are representative of possible future climates. [10] study how exogenous stochastic processes (e.g., meteorological events) influence building thermal processes, and how endogenous (building-internal) process knowledge (e.g., occupancy patterns) can lead to improved building operation.

3 Simulation Framework

We consider an optimisation framework in which our task is to optimise an objective function \mathcal{J} subject to

a set of constraints over the model, $\chi(\Phi)$. For example, we may want to define an optimal controller for controlling the heating system in a building.

We assume that the model Φ_P that we are simulating requires a set of inputs generated by an exogenous stochastic process Φ_O . For example, in building energy simulation, a model Φ_P consists of the building itself, e.g., the building envelope with internal zones, climate control equipment such as HVAC and sensors/actuators, etc.

We partition the variables in a Modelica model Φ_P as $\mathcal{V} = \mathcal{V}^P \cup \mathcal{V}^O$, where \mathcal{V}^P denotes the endogenous variables and \mathcal{V}^O denotes the exogenous variables. Endogenous variables \mathcal{V}^P do not depend (at least, not directly) on any exogenous stochastic process: at each simulation step, they are deterministically calculated by the Modelica solver. By contrast, exogenous variables \mathcal{V}^O depend directly on exogenous inputs, which change over time due to the exogenous stochastic process Φ_O . Therefore, the values of \mathcal{V}^O must be updated every time the stochastic process Φ_O produces new inputs.

For example, the exogenous variables might be weather variables that provide a set of input conditions for weather for a Modelica simulation of Φ_P . In fact, the existing building library [13] has inputs for up to 30 weather variables, such as temperature, wind-speed, etc.

Figure 4 shows an example of a discrete-time simulation process with exogenous model inputs \mathcal{V}_t^O at each time step t . The exogenous model Φ_O performs inference independent of the Modelica simulation, and provides an input for variables \mathcal{V}_t^O at each time step. The Modelica simulation uses these inputs to conduct its simulation. A key insight into this process is that the Modelica model Φ_P must provide inputs of its endogenous variables \mathcal{V}_t^P to the simulation at time $t + 1$, since the model would otherwise take (incorrect) default values for \mathcal{V}_{t+1}^P .

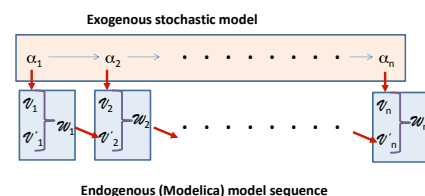


Figure 1: Simple schematic of simulation process with exogenous model inputs

We formalise this process as follows. We define our simulation system as consisting of two models: (a) an

exogenous (possibly stochastic) model Φ_O with variables α , of which a subset \mathcal{V}^O are output variables; (b) a deterministic (endogenous) model Φ_P with variable set \mathcal{V} partitioned into input variables \mathcal{V}^O and internal variables \mathcal{V}_P , and parameters Θ (which are constants over a simulation).

3.1 Stochastic Model Analysis

We further assume an exogenous model \mathcal{E} defined by $\Theta = \psi(\alpha)$ that generates the parameter assignment $\hat{\theta}$. If \mathcal{E} is stochastic, then we have $Pr(\Theta) = Pr(\alpha)$, which defines the joint distribution over α .

We assume a two-step process for model analysis. First, we perform exogenous analysis, which takes the joint set of stochastic inputs α , and through Monte-Carlo (MC) sampling, generates an ensemble of predictions for the parameter set α . Second, we run a simulation for each element of the ensemble, generating an ensemble \mathcal{S} of simulation outputs. Finally, we perform some analysis of the ensemble \mathcal{S} to compute our objective.

3.2 Stochastic Simulation Process

[5] define a *good* probabilistic simulation/forecast as the process of maximizing the accuracy of the predictive distributions subject to calibration, where *accuracy* refers to the spread of the predictive distributions, and is a property of the forecasts only. *Calibration* is the statistical compatibility between the predictive simulation output (or distributions for stochastic models) and the observations. This is a joint property of the forecasts and the observations. We can jointly assess calibration and accuracy by using proper scoring rules, such as the logarithmic score or the continuous ranked probability score [6].

For example, a proper scoring rule is a function $s(\zeta, x)$ that assigns a numerical score to each pair (ζ, x) , where ζ is the predictive distribution and x is the verifying observation.

3.3 Simulation Analysis

Given a set of n possible input streams to Φ_P , we run n simulations. The key is to now use these n simulations to solve our tasks in order to optimise \mathcal{J} .

Consider the case where we aim to compute an optimal control that optimises \mathcal{J} . Given the n simulations, we want to compute a robust control u^* .

Robust control methods are designed to function properly (e.g., maintain stability) under the condition

that uncertain parameters or disturbances are within some (typically compact) set. For example, this may include the assumption of bounded modelling errors. In contrast with adaptive control (which can adapt to changes in environmental conditions or measurements), robust control methods are static.

In our case, we assume that the MC sampling provides a statistically sound set of simulation conditions. Given that, we can either optimise the worst-case outcome, or optimise within the bounds to the input ensemble.

4 Application Domain: Energy Modeling

4.1 Building Simulation

For the analysis and prediction of the dynamic behavior of building performance indicators such as energy consumption and thermal comfort, building performance simulation (BPS) is a key enabling technology. Previous work has shown that the use of BPS is mostly limited to building design and for checking code compliance for the detailed design [8].

BPS makes a number of assumptions that violate realistic building characteristics. For example, almost all BPS model variables are assumed to be deterministic, even though they are uncertain, due to uncertainty in material characteristics and to external and internal condition changes over time. For example, a BPS model contains a range of internal parameters that are only known imprecisely, e.g., wall/ floor/ ceiling heat-transfer parameters. In addition, this type of model behaves differently based on the building occupancy and usage, both of which change over time.

4.2 Incorporating Weather Forecasts

Today, the preferred method of probabilistic weather prediction is based on ensembles of Numerical Weather Prediction (NWP) forecasts. In this case, each ensemble member is a single-valued, deterministic forecast from an NWP model, i.e., a simulation of an NWP model. The forecasts differ from each other with respect to the two major sources of uncertainty: (1) initial conditions and/or (2) model formulation. Figure 2 shows an example of an ensemble of 11 pressure predictions over time.

The ensemble of forecasts must be post-processed in order to provide an interpretable, single forecast. In other words, statistical post-processing aims to gener-

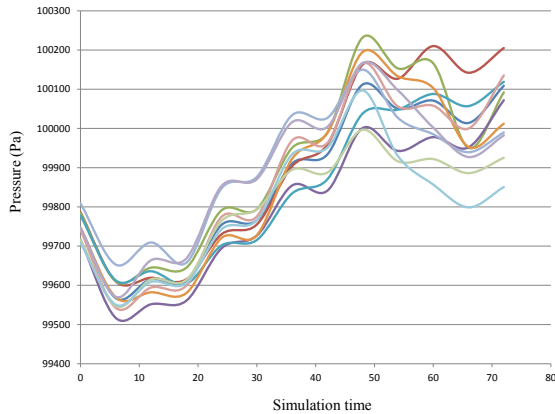


Figure 2: Weather ensemble of 11 pressure predictions over time

ate a calibrated, sharp predictive distribution from the output of NWP ensembles. Two general approaches to the statistical post-processing of forecast ensembles have emerged, namely

- Bayesian model averaging (BMA) [7], where each ensemble member is associated with a kernel function, with a weight that reflects the member’s relative accuracy.
- ensemble model output statistics (EMOS) [4] or nonhomogeneous Gaussian regression (NGR), which fits a single, parametric predictive PDF using summary statistics from the ensemble.

Consider an ensemble forecast, $\lambda_1, \dots, \lambda_m$, for surface temperature, T , at a given time and location. BMA employs Gaussian kernels with a linearly bias-corrected mean: the BMA predictive PDF is the Gaussian mixture with mean \mathcal{N} and variance σ^2 .

$$p(T|\lambda_1, \dots, \lambda_m) = \sum_{i=1}^m w_i \mathcal{N}(a_i + b_i \lambda_i, \sigma^2),$$

with the BMA weights w_1, \dots, w_m , bias parameters a_1, \dots, a_m and b_1, \dots, b_m , and a common spread parameter σ^2 .

The major drawback to this current ensemble approach to physical simulation is that it only applies to single variables, at single locations and single look-ahead times. A key objective in this area is to compute physically consistent probabilistic forecasts of spatio-temporal simulation trajectories.

4.3 Example: Underfloor Heating Example

Consider the case where we can compute a control setting for the underfloor heating in a zone Z , where

we have uncertainty over the weather forecast and the occupancy for the following day.

We apply our approach to the optimisation of underfloor heating control. Our task is to compute the time interval I during which we “charge” (or heat up) the underfloor slab during the night, such that we jointly maximise user comfort (U_c) and minimise energy usage (U_e) over the following day. Figure 3 depicts a simple example of an underfloor heating system for a house.

We can formulate this task by defining \mathcal{J} as the weighted sum of user comfort and energy usage, with corresponding weights w_c and w_e :

$$\mathcal{J} = w_c U_c + w_e U_e \quad (1)$$

subject to

$$U_c \geq U_c^*, U_E \geq 0 \quad (2)$$

$$\chi(\Phi) \quad \text{are satisfied} \quad (3)$$

Figure 3: Simple schematic of underfloor heating system for a house

4.4 Underfloor Heating with Stochastic Forecasting

This section describes our underfloor heating model that incorporates stochastic forecasts for weather and occupancy. Figure 4 depicts the variation in tempera-

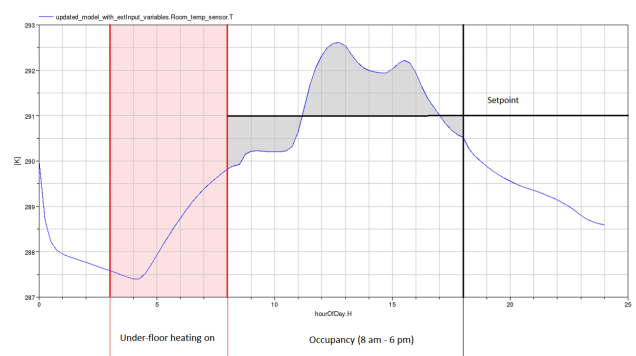


Figure 4: Simulation process for under-floor heating system. The red area shows the hours during which the heating is on. The gray area shows the difference between setpoint and room temperature during office hours.

ture over a day, given that the underfloor heating system is switched on for the period 3–8 am. In this example the temperature set-point for the day is 18° Celsius,

and our objective is to maintain this temperature as closely as possible, in order to optimise the occupants' comfort. The gray area between the actual temperature during the day and the set-point is used to compute a discomfort index, i.e., it is the area denoting the failure to maintain the set-point.

We employ three different models for this application:

- a stochastic model for weather variable prediction;
- a stochastic model for occupancy prediction;
- a Modelica model for simulating the occupied zone in a building with underfloor heating, given as inputs the weather forecasts and the predicted occupancy.

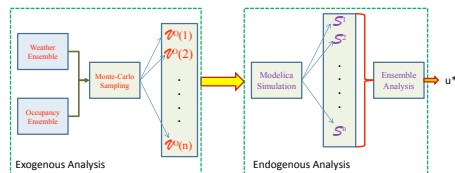


Figure 5: Computational architecture for analysis of underfloor heating system.

Figure 5 depicts our computational architecture, showing the two phases of exogenous computation, where we generate ensembles for weather and occupancy forecasts, and endogenous computation, where we create an ensemble of simulations based on the input ensembles, and then compute the control output u^* that optimises our objective function \mathcal{J} , given the simulation ensembles.

5 Implementation

We have partially implemented the computational architecture described in the previous section. In this section we provide implementation details on our energy simulation model and its inputs, as well as how we intend to use the model for computing an optimal control action u^* .

5.1 Room model

We model a room of one of the buildings on our university campus. This room is an open-space office with a maximum capacity of 12 occupants. The room is equipped with typical office furniture (desks, computers, printers, etc.). The only heating system is

under-floor heating. Additionally, the room has 8 windows and 2 doors. The room is also equipped with sensors that monitor temperature, presence, and luminance.

We model this room by using the Buildings library, developed by Wetter et al. [13]. Figure 6 contains a graphical representation of our model. The main components are:

1. a room component, which extends Buildings.Rooms.MixedAir;
2. an external weather file;
3. heat gains based on occupancy and equipment in the room;
4. an under-floor heating component.

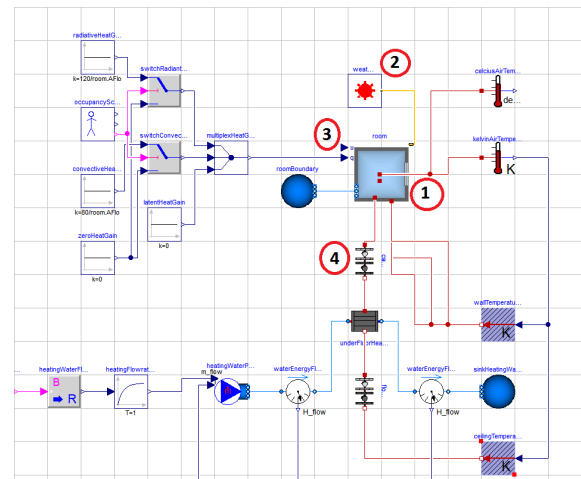


Figure 6: Room model with under-floor heating

5.2 Weather forecast ensembles

Weather is one of the main inputs to our model. Weather data can be either from the past (historical weather records) or in the future (weather forecasts). Since our objective is to implement a control framework, we are interested in weather forecasts. In this section we discuss how we obtain and process weather forecasts.

As mentioned previously, probabilistic weather forecasts are based on ensembles. These ensembles are generated routinely by various data centers around the world. In particular, we use weather forecasts generated by the Global Ensemble Forecast System (GEFS) model [2], which is developed and run by the National Oceanic and Atmospheric Administration (NOAA) in the United States.

As the name suggests, the GEFS is a global model, i.e. it produces forecasts for the whole planet. These forecasts are available for download free-of-charge from the NOAA file servers. The GEFS model produces forecasts up to 16 days in advance; however, since the accuracy tends to degrade quickly, we consider only the first 7 days of prediction. For these first 7 days, the model provides a spatial resolution of 1 degree latitude by 1 degree longitude, and a temporal resolution of 6 hours. The GEFS produces 20 ensemble members. Each member contains the trajectories of various weather variables, e.g. temperature, humidity, pressure, etc.

Our goal is to use these ensemble forecasts to generate probabilistic weather inputs for our model. In order to accomplish this, after downloading the forecast files, we need to carry out a series of steps:

1. forecasts must be spatially interpolated to the point of interest;
2. forecasts must be temporally interpolated;
3. the weather variables that are relevant to our model must be extracted from the forecasts; additionally, some weather variables required by the model are not directly included in the forecasts (e.g. direct and diffuse solar radiation), and therefore must be calculated from the information that is available;
4. the extracted and calculated variables must be statistically post-processed, in order to provide probability distributions;
5. the probability distributions calculated above must be sampled (e.g. by using Monte Carlo methods) to provide weather scenarios;
6. finally, for each sampled scenario, a weather file in the format required by the model must be produced.

We have developed software that performs the above steps, with the exception of the statistical post-processing. At the moment of this writing, instead of generating probability distributions and then sampling from those, we simply create 20 different weather files for each of the 20 ensemble forecasts generated by the GEFS; then, we provide these weather files as inputs to our model. Figure 7 depicts the steps we have implemented to provide weather input to our model.

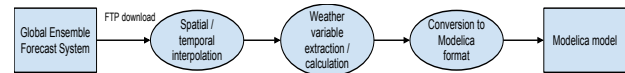


Figure 7: Steps to provide weather input to the room model

5.3 Sensitivity Analysis

The Modelica Buildings library accepts weather files specified as tables of n rows and 30 columns. Each row contains weather conditions for a specific time step, and each column contains the values of a weather variable, specified as real numbers. However, each weather variable has a different impact on the model output (i.e., the room temperature): some variables affect the room temperature more strongly than others; some variables do not affect the room temperature at all. Hence, it is important to precisely assess the impact of each weather variable on the model output, so that only relevant variables need to be extracted from the forecasts.

In this context, we have performed sensitivity analysis on our model in the following way. First, we ran a baseline simulation with a weather file containing historical data. Then, we altered each weather variable in the file by increasing and decreasing its values by 10%, 20% and 30%. Each variable was altered independently of the others; i.e., when we altered one variable, all other variables retained their original values.

For each weather variable, we generated 6 simulations corresponding to the variations in the range of $\{-30\%, -20\%, -10\%, 10\%, 20\%, 30\%\}$. We measured the error between the baseline and each variation; the error was calculated as the integral of the difference of room temperature. The error provides an indication of how much a variable affects the simulation output, with higher error corresponding to higher sensibility.

The results indicate that the model is most sensitive to the following weather variables:

1. dry bulb temperature;
2. direct, diffuse and global solar radiation;
3. opaque sky cover;
4. wind speed and direction;
5. dew point.

Therefore, only these variables are extracted from the GEFS weather forecasts and provided to the model.

5.4 Simulation process

We use the model to address the task of optimal under-floor heating control. Since under-floor heating is a slow-response system, it is normally turned on at night: it is at this time that the concrete slabs are “charged” with heat, which will then be released in the room over the following day. In this context, the output of the control task is u^* , i.e. the amount of hours during which the under-floor heating will be turned on.

In order to calculate u^* we simulate 9 different scenarios S_u , where we vary the amount of under-floor heating hours u from 0 to 8. For each S_u , we calculate $\mathcal{J}_u = w_c U_c + w_e U_e$. Finally, we calculate $u^* = u_{\arg \max} \{ \mathcal{J}_u \}$. The value u^* is then given as input to a Building Management System (BMS) which opens the under-floor heating valves for the required amount of time.

This process is repeated every day, for instance at 10 pm. In other words, every 24 hours we run a new series of simulations and we calculate a new control action u^* based on weather (and, potentially, occupancy) forecasts for the day after. However, each time we run a new series of simulations, we cannot reset the model variables to pre-defined initial values. In fact, as stated in section 3, the endogenous variables must be initialized with the values of the previous simulation, whereas the exogenous variables must be initialized according to the external stochastic processes. Using Dymola, this means that the *dsin.txt* file (which provides initial values to variables) must contain the final values of the simulation which generated u^* 24 hours before. This process is depicted in figure 8.

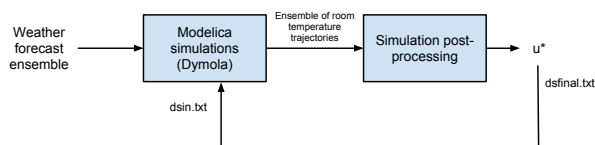


Figure 8: Steps to compute u^* every 24 hours. The post-processing step contains the logic to calculate u^* . The file *dsfinal.txt* contains the final values of the previous simulation which generated u^* .

5.5 Preliminary control results

In order to test our control approach, we first applied it to historical weather data. Our goal here is to compare our control strategy with a 5-hour fixed-schedule strategy, in terms of user comfort and energy usage. We use the simulation process described in the previ-

ous section; the only change is that we use historical weather data instead of forecasts.

We use a set-point of 23 degrees Celsius for week days, and 16 degrees Celsius for weekends. The fixed schedule strategy operates the under-floor heating for 5 hours every night, between 3 am and 8 am. Moreover, it does not differentiate between weekdays and weekends¹. Our control strategy, instead, tries to minimize the error between set-point and room temperature, and thus will tend to turn off the under-floor heating during weekends, when the set-point is lower.

Figure 9 shows the average room temperature obtained with our control strategy (blue trajectory) and the fixed schedule strategy (red trajectory). Although there is some amount of error for both strategies, it is clear that, on average, our control strategy performs better, i.e. it is closer to the desired set-point. Possible ways to further improve our control strategy might consist in (1) leaving the set-point unchanged over weekends (thus avoiding the cooling down of building materials, at the expense of higher energy usage), and (2) extending the possible number of under-floor heating hours to 9 or 10 (at the moment we keep the maximum number of hours to 8).

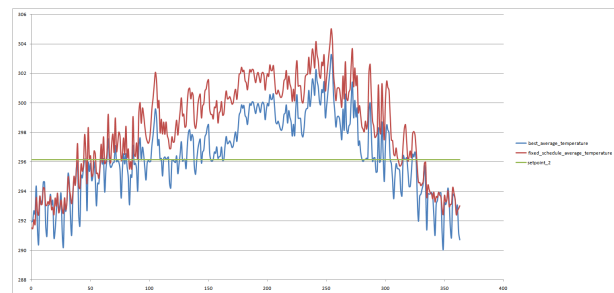


Figure 9: Comparison of average temperature obtained with our control strategy (blue) and a 5-hour fixed schedule strategy (red). The red line represents the desired set-point. The horizontal axis represents days and the vertical axis represents degrees Kelvin.

Figure 10 compares the energy usage of the two strategies. It is apparent that, over the whole year, our control strategy requires significantly less energy than the fixed schedule strategy. This is mainly due to (1) savings during weekends, and (2) savings during the summer season, when the under-floor heating is not needed. It is worth mentioning that, within the model, the energy usage is calculated as the amount of energy (in Joule) that is required to heat up the water which will flow into the under-floor heating pipes.

¹It is worth noting that this control strategy was actually implemented on the building on our university campus.

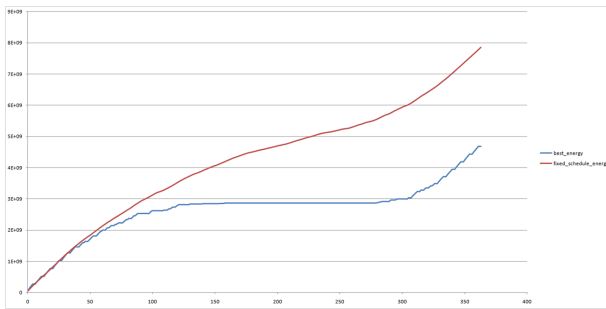


Figure 10: Comparison of energy consumed by our control strategy (blue) and a 5-hour fixed schedule strategy (red). The horizontal axis represents days and the vertical axis represents energy in Joule.

5.6 Adding stochastic inputs to the control framework

The preliminary implementation discussed in the previous section did not include stochastic inputs. In fact, both weather and occupancy are assumed to be deterministic processes over a day. In this section, we discuss how we intend to extend this control framework in order to include stochastic inputs.

As explained in section 3.1, given an exogenous model Φ_O with variables α , we first compute the joint probability distribution $Pr(\alpha)$, and then we generate an ensemble of predictions for α through Monte Carlo sampling; the predictions will then be used as exogenous inputs to the Modelica model Φ_P .

In our application domain, the exogenous model Φ_O is a combination of stochastic weather and occupancy. Therefore, each prediction p for Φ_O will contain the trajectories of weather variables, plus the number of occupants in the room at each time t . In order to use predictions p as exogenous inputs to our control framework, we use algorithm 1. This algorithm computes u^* by searching through a search space composed of $9 \times n$ simulations, where n is the number of predictions for α generated through Monte Carlo sampling.

It is significant to note that the simulated room temperature can change significantly on the basis of different predictions for α . Figure 11 shows an ensemble of 5 room temperature trajectories, obtained with 5 different members of a weather forecast ensemble. Given this significant variability, we believe that using a combination of stochastic weather and occupancy prediction could yield better results than using deterministic forecasts.

Algorithm 1 Algorithm to compute a control action based on an ensemble of exogenous predictions.

for u between 0 and 8 **do**

for each prediction p **do**

calculate $error_u^p$

calculate $energy_u^p$

$J_u^p \leftarrow w_c error_u^p + w_e energy_u^p$

end for

$J_u \leftarrow \sum_{p=1}^n J_u^p$

end for

return u^* for which J_u is minimized

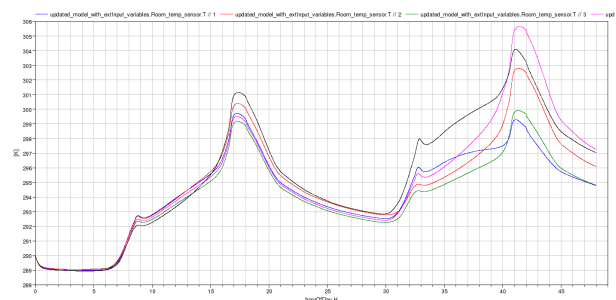


Figure 11: Ensemble of 5 room temperature trajectories, generated with 5 members of a weather forecast ensemble. The horizontal axis contains hours and the vertical axis contains degrees Kelvin.

6 Discussion

We have described an approach to extend Modelica simulation with multiple ensembles generated by exogenous stochastic simulations. This approach couples a discrete-time stochastic simulation with a Modelica simulation, in which the stochastic simulation generates an input to the Modelica model for each time step. Further, the system state from the Modelica model for time t must be used to initialise the model at time $t + 1$. This methodology can enable Modelica to be used for optimisation, and for embedded control and optimisation applications.

Although this approach works well for slower systems, for fast systems (where each time step is small) the computational overhead of initialising a Modelica simulation for each time step hinders real-time and embedded applications. This exposes the limitation of Modelica in two ways: (1) the lack of an in-built stochastic modeling capability; and (2) the inability to accept inputs (e.g., from sensors and actuators) during a simulation. We argue that, in order to gain acceptance for real-world applications, Modelica must ex-

tend its language and computational tools to incorporate methods for dealing with these two deficiencies. Bouskela et al. [1] propose a language extension to partially deal with the first deficiency, but further work is necessary.

References

- [1] Daniel Bouskela, Audrey Jardin, Zakia Benjelloun-Touimi, Peter Aronsson, and Peter Fritzon. Modelling of uncertainties with Modelica. In *Proceedings of the 8th International Modelica Conference*, Dresden, Germany, 2011. Linköping University Electronic Press.
- [2] R. Buizza, P. L. Houtekamer, Gerald Pellerin, Zoltan Toth, Yuejian Zhu, and Mozheng Wei. A comparison of the ecmwf, msc, and ncep global ensemble prediction systems. *Monthly Weather Review*, Vol. 133, No. 5, 2005.
- [3] M. Eames, T. Kershaw, and D. Coley. A comparison of future weather created from morphed observed weather and created by a weather generator. *Building and Environment*, 2012.
- [4] H.R. Glahn and D.A. Lowry. The use of model output statistics (mos) in objective weather forecasting. *Journal of Applied Meteorology*, 11(8):1203–1211, 1972.
- [5] T. Gneiting, F. Balabdaoui, and A.E. Raftery. Probabilistic forecasts, calibration and sharpness. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 69(2):243–268, 2007.
- [6] T. Gneiting and A.E. Raftery. Strictly proper scoring rules, prediction, and estimation. *Journal of the American Statistical Association*, 102(477):359–378, 2007.
- [7] J.A. Hoeting, D. Madigan, A.E. Raftery, and C.T. Volinsky. Bayesian model averaging: a tutorial. *Statistical science*, pages 382–401, 1999.
- [8] C.J. Hopfe and J.L.M. Hensen. Uncertainty analysis in building performance simulation for design support. *Energy and Buildings*, 2011.
- [9] D. Jacob, S. Burhenne, A.R. Florita, and G.P. Henze. Optimizing building energy simulation models in the face of uncertainty, 2010.
- [10] Y. Jiang and T. Hong. Stochastic analysis of building thermal processes. *Building and Environment*, 28(4):509–518, 1993.
- [11] B. Johansson and P. Krus. Probabilistic analysis and design optimization of modelica models. In *Paper presented at the 4th International Modelica Conference*, 2005.
- [12] I.A. Macdonald. *Quantifying the effects of uncertainty in building simulation*. PhD thesis, Department of Mechanical Engineering, University of Strathclyde, 2002.
- [13] M. Wetter. A modelica-based model library for building energy and control systems. In *Proc. of the 11th IBPSA Conference*, 2009.

A Toolchain for Real-Time Simulation using the OpenModelica Compiler

Niklas Worschech Lars Mikelsons
Bosch Rexroth
Rexrothstraße 3, 97816 Lohr am Main

Abstract

Nowadays, simulation is the key technology to shorten development times, while increasing the functionality of products. In this context simulation is always used in order to verify characteristics of the product under consideration. In the past simulation was mostly done offline, i.e. not synchronized to real-time. Due to the increased computing power, the relevance of real-time simulation has increased in the last years. Therefore, several simulation environments offer a toolchain for real-time simulation, e.g. the Real-Time Workshop integrated in Simulink. In this paper such a toolchain (although not yet fully automated) for the OpenModelica Compiler (OMC) is presented using a hydro-mechanical system as an example. Thereby, this paper describes a modular C++ Simulation-Runtime for the OMC including a numerical integration method suitable for real-time simulation as well as modeling details of the example system using Modelica. *Keywords: real-time; simulation; runtime; OpenModelica*

1 Introduction

Simulation is always based on models. These models can be mind-models, scaled physical models or mathematical models. No matter what kind of model is used, the purpose of simulation is mostly the validation of characteristics of physical systems. Nowadays, even detailed mathematical models can be simulated in relatively short time. Hence, computer-simulation is an important tool in the mechatronic development cycle and helps to reduce costs by shorten the development process. The mechatronic-development cycle involving the validation process is visualized in the V-Model in figure 1.

Clearly, the level of detail of the employed model plays a very important role. To obtain a model with a higher level of detail, more modeling effort has to be invested and one has to expect longer simulation times.

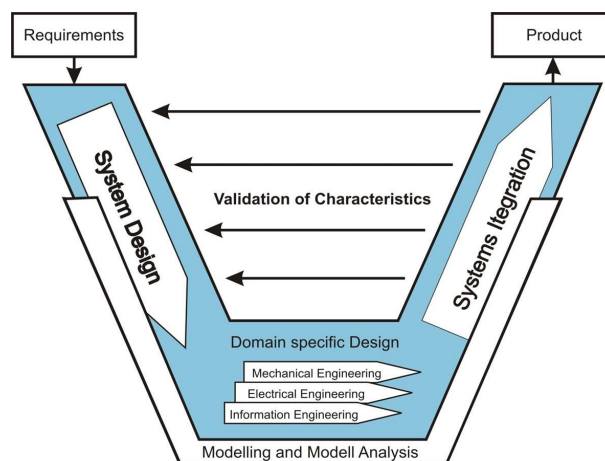


Figure 1: V-Model of the mechatronic development cycle

A proper model is as simple as possible, but still complex enough to reproduce the physical effects under consideration [9]. However, there exist tasks that can not be fulfilled satisfactorily with the help of non-real-time simulations regardless of which level of detail is used. These are among others:

- Setting up Simulators (e.g. driving simulator),
- Controller testing,
- Physical Component testing.

Real-time simulation refers to a mathematical model of a physical system including a numerical integration method that can execute at the same rate as actual "wall clock" time. Hence, using real-time simulation, the real system can be replaced by a virtual system which makes real-time simulation suitable for the applications mentioned above. Due to this possibility and the increased available computing power, real-time simulation became very popular in the recent years.

Consequently, many commercial simulation tools offer a complete toolchain for real-time simulation. Such

a toolchain consists of a modeling environment, a simulation-runtime and a compiler which can compile the model for a real-time-target. Simulink together with the Real-time Workshop form the toolchain offered by The MathWorks. Some other tools do not offer an own compiler, but an export to Simulink, so that the real-time Workshop can be used. There are also tools which offer an integrated solution. However, currently the OMC lacks such an automated toolchain at all. In this paper a C++ Simulation-Runtime is presented which forms the basis for a toolchain for real-time simulation. This modular C++ Simulation-Runtime contains a numerical integration method suitable for real-time simulations of hydraulic systems and can also be used for co-simulation.

This contribution is structured as follows. In section 2 the C++ Simulation-Runtime and its structure is presented. After that the toolchain for real-time simulation is explained using an application example in section 3. Here, the C++ Simulation-Runtime is compiled together with the application example for the real-time operating system Scale-RT [2] and executed on a real-time-target after that. The paper closes with a conclusion and an outlook.

2 A C++ Simulation-Runtime for OpenModelica

In order to set up an automated toolchain for real-time simulation, a new C++ Simulation-Runtime was designed. The design-guidelines were chosen to obtain a simulation-runtime that is easy to

- maintain,
- extend,
- configure.

Therefore, it is much easier to add new numerical integration methods, extend its functionality with new algorithms (e.g. for initialization) or just to fix bugs. In order to obtain a simulation-runtime that realizes these design-guidelines, the solver-component which implements the numerical integration method is separated from the system-component which represents the system of differential-algebraic equations (DAE). Note, that this design is completely contrary to the idea of inline-integration which was invented in order to increase the computational efficiency [8]. In the next section a general overview is given. After that the Event-Handling strategy is explained. In section 2.4

the chosen numerical integration method for real-time simulation is described.

2.1 Components Overview and General Interface Description

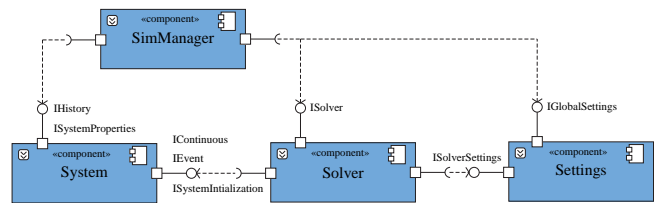


Figure 2: Components of the C++ Simulation-Runtime

In figure 2 the component diagram of the C++ Simulation-Runtime is pictured. The solver-component consists of a set of integration methods, e.g. CCode from the Sundials library [12]. The SimManager-component controls the simulation. Besides standard-tasks like starting and stopping of the simulation, the SimManager is able to synchronize different systems and solvers and hence allows for co-simulation. The settings-component is used to configure the simulation, e.g. set solver-tolerances. The system-component represents the DAE and therefore includes the Modelica-System class. This class is generated by a new code-generation module inside the OpenModelica compiler [10]. As mentioned above the solver-component is separated from the system-component and thus interfaces are used (see figure 3).

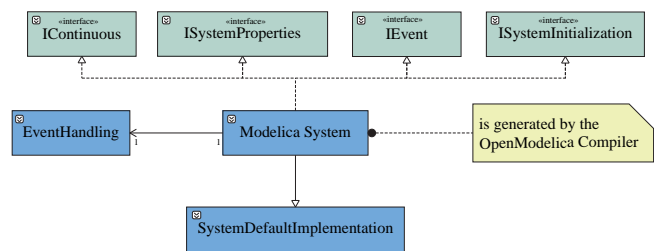


Figure 3: Modelica-System class

The C++ Simulation-Runtime is able to handle systems with a lot of different properties as shown in figure 4. Some of the properties (like *isAlgebraic*) are standard properties and used to automatically select a suitable numerical solution method for the corresponding system. Other properties are not yet reported by the OMC to the C++ Simulation-Runtime. A flag to use a symbolic jacobian for the numerical integration is part of current work. The generation of the symbolic

jacobian is described in [5]. The interface *ISystemI-*

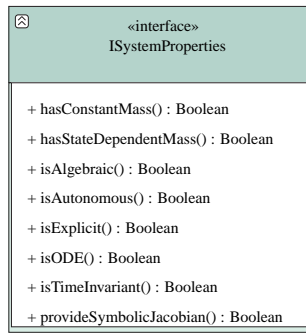


Figure 4: ISystemProperties Interface

*nitia*lization is used to initialize the Modelica-System at the beginning of the simulation. Since the efficient initialization of models is part of current work [6], the currently implemented algorithms are rather basic. However, due to the design of the C++ Simulation-Runtime, new initialization-algorithms can be easily added. The communication between solver and sys-

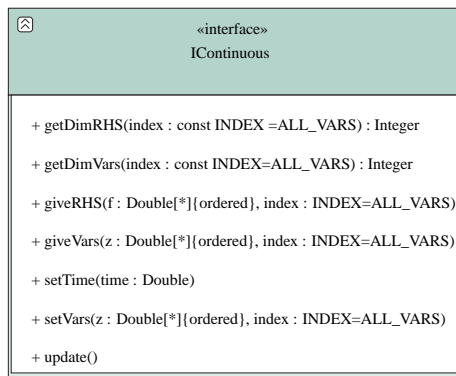


Figure 5: IContinuous Interface

tem is defined by the interface *IContinuous* (see figure 5). The method *giveVars* returns the state-vector **z**. The state-vector is sorted according to the variable-index (see table 2.1) and hence it is possible to access a corresponding part of the state-vector by passing the variable-index. This sorting allows for efficient generation of the jacobian [11]. The remaining methods

Variable Index	Description
VAR_INDEX0	States of systems of 1st order
VAR_INDEX1	1st order States of systems of 2nd order, e.g. positions
VAR_INDEX2	2nd order States of systems of 2nd order, e.g. velocities
DIFF_INDEX3	Constraints on position level only
DIFF_INDEX2	Constraints on velocity level only
DIFF_INDEX1	Constraints on acceleration level only
ALL_RESIDUALS	All constraints
ALL_STATES	
ALL_VARS	

Table 1: The Variable Index

are basic methods needed for the numerical integra-

tion process.

In case that the OMC returns algebraic equation

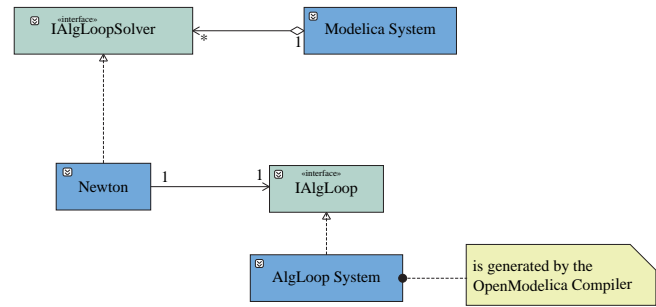


Figure 6: Solving Non Linear and Linear Systems

systems (as shown in figure 6), an instance of the AlgLoop-System class is created for each equation system. Once again, the Algloop-System class provides a method which allows to choose an adequate numerical solution method.

The simulation results are currently stored in a tabulator separated text-file. The Modelica-System class uses an instance of type *IHistory* to store the simulation results. Moreover, the storing instance uses a policy class for the implementation of the storing behavior [3]. This allows an extension of the output mechanism of simulation results, e.g storing the results in a buffer for further processing. In the future simulation results will be stored in the new Modelica result-file-format.

2.2 Integration Loop

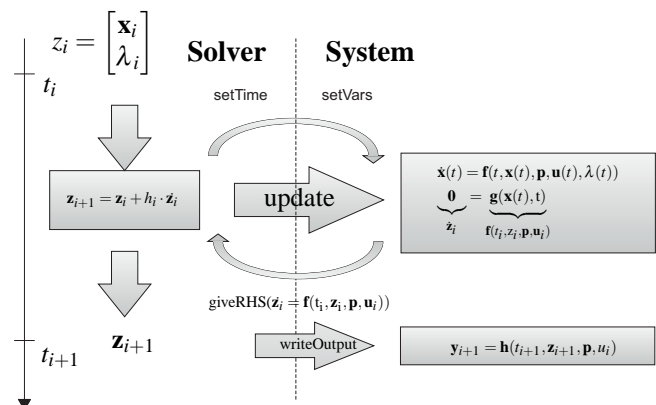


Figure 7: Integration loop in the C++ Simulation-Runtime

A scheme of the integration loop for a semi-explicit DAE

$$\dot{\mathbf{x}}(t) = \mathbf{f}(t, \mathbf{x}(t), \mathbf{p}, \mathbf{u}(t), \lambda(t)), \quad (1a)$$

$$\mathbf{0} = \mathbf{g}(\mathbf{x}(t), t), \quad (1b)$$

can be seen in figure 7. Here, \mathbf{x} denotes the states, λ is the vector of algebraic variables, \mathbf{p} are the parameters and $\mathbf{u}(t)$ are the system inputs. The time-step starts by setting the previously calculated state-vector and the current time. The right-hand-side of equation 1a is evaluated by calling *update*. Note that algebraic loops are solved within this call. After that *giveRHS* gives the right-hand-side to the numerical integration method which performs the integration step (e.g. using Forward-Euler).

2.3 Event-Iteration

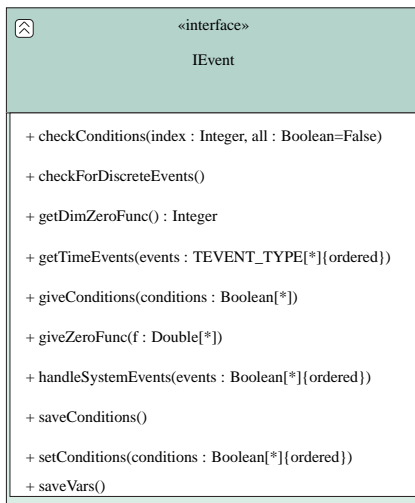


Figure 8: IEvent Interface

To handle discontinuities the Modelica-System implements the *IEvent* interface (figure 8). For each continuous event from the Modelica model, a zero-crossing- function and a corresponding condition variable is created. Thereby, the zero-crossing-functions are interpreted as transitions in a state-graph. To be more precise, the zero-crossing-functions are always negative as long as no event occurs. A positive zero-crossing-function indicates an event and in the consequence the event is handled (and the event-iteration is started) such that the corresponding zero-crossing-function is negative again. Note, that this is fundamental difference to the treatment of events in the current C Simulation-Runtime and allows the use of the built-in zero-detection algorithms of the Sundials library. These algorithms are very efficient since all ODE/DAE solvers of the Sundials library are multi-step methods and hence the solution polynomial is at hand with no additional effort.

When a zero is found an event-iteration is started as pictured in figure 9. The input of the event-iteration

is an event-vector \mathbf{e} indicating which zero-crossing-function (i.e. transition) is active. The relevant relation expressions are evaluated and stored in a condition-vector using *checkConditions*. This condition-vector is used in the *update* method to evaluate the right-hand-side of equation 1a. The method *saveVars* is called to save the predecessor values of all variables.

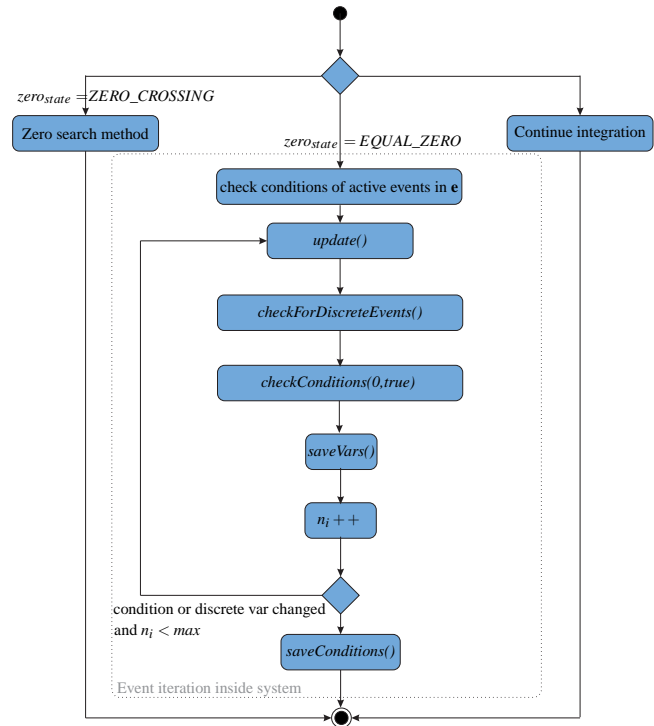


Figure 9: Event Iteration within an integration step

2.4 Real-time Simulation

Real-time Simulation refers to a mathematical model of a physical system including a numerical integration method that can execute at the same rate as actual "wall clock" time. Hence, two requirements have to be met:

- The simulation has to be faster than the "wall clock" time.
- A predictable worst-case runtime is required.

The first requirement is a requirement on the computational complexity and hence a requirement for the model as well as for the numerical integration method. An approach for the generation of models suitable for real-time simulation can be found in [13]. The choice of the numerical integration method is even more restricted by the second requirement which is mostly harder to meet than the first one. A predictable worst

case runtime can only be obtained with non-iterative algorithms. To be more precise implicit numerical integration methods can not be used (without modification) inside a real-time process. Note that this requirement is rather problematic in the context of stiff ODEs and DAEs. Furthermore, step-size control produces a non-predictable runtime and can thus also not be used. The same holds for many algorithms for the detection of zero-crossings.

Since explicit numerical integration methods are not suited for many practical problems and implicit methods are not allowed inside a real-time process, linear-implicit integration methods with fixed step size are very common for real-time simulation [4]. Using a linear-implicit integration method, not a non-linear, but a linear system of equations has to be solved. This operation can be performed with an upper bound for the computational effort and hence linear-implicit integration methods can be used in real-time processes. Linear-implicit methods can for example be obtained by linearizing the numerical integration method. In that case linear-implicit methods inherit the stability properties of the corresponding implicit method due to the linearity of Dahlquist's test equation [11].

The most popular linear-implicit integration scheme is the linear-implicit Euler-method due to its simplicity and stability properties, i.e. it is A- and L-stable like the Backward-Euler [7]. These properties make it better suited for practical (i.e. stiff) problems than explicit methods. Unfortunately, it is of the same order as Backward-Euler which might be problematic in combination with a fixed-step size for low tolerances. An alternative is the linear-implicit trapezoidal-rule. This method has the same complexity as Backward Euler but is of order two. However, the linear-implicit trapezoidal rule is not L-stable due to the stability properties of the trapezoidal-rule and should thus not be used for stiff problems.

The C++ Simulation-Runtime offers an A- and L-stable linear-implicit integration method of order three which will be called LI3 in the following. This method was designed for the solution of discretized unsteady incompressible Navier-Stokes equations originally and has not been used for real-time simulation yet (to the author's knowledge) [14]. For an ODE as

in equation 1a the method can be written as

$$\mathbf{k}_1 = \mathbf{x}_n + \frac{2h}{3} \mathbf{L} \cdot \mathbf{f}(\mathbf{x}_n, t_n), \quad (2)$$

$$\mathbf{k}_2 = \mathbf{L}(\mathbf{x}_n - \frac{h}{2} \mathbf{J} \cdot \mathbf{k}_1 + \frac{h}{3} \mathbf{f}(\mathbf{x}_n, t_n) + \frac{h}{3} \mathbf{f}(\mathbf{k}_1, t_n + \frac{2h}{3})), \quad (3)$$

$$\bar{\mathbf{k}} = \frac{9}{4} \mathbf{k}_1 - \frac{3}{4} \mathbf{k}_2 - \frac{1}{2} \mathbf{x}_n, \quad (4)$$

$$\mathbf{k}_3 = \mathbf{L}(\mathbf{x}_n - \frac{h}{2} \mathbf{J} \cdot \bar{\mathbf{k}} + \frac{h}{4} \mathbf{f}(\mathbf{x}_n, t_n) + \frac{3h}{4} \mathbf{f}(\mathbf{k}_1, t_n + \frac{2h}{3})), \quad (5)$$

$$\mathbf{x}_{n+1} = \mathbf{L}(\mathbf{x}_n - \frac{h}{2} \mathbf{J} \cdot \bar{\mathbf{k}} + \frac{h}{4} \mathbf{f}(\mathbf{x}_n, t_n) + \frac{3h}{4} \mathbf{f}(\mathbf{k}_2, t_n + \frac{2h}{3})), \quad (6)$$

where

$$\mathbf{L} = (\mathbf{E} - \frac{h}{2} \mathbf{J})^{-1}. \quad (7)$$

Here \mathbf{J} denotes the jacobian of \mathbf{f} (or at least an approximation) and h is the step-size. Thus, one time-step requires three evaluations of the right-hand side of the ODE. Moreover, four linear systems of equations of the same dimension as \mathbf{x} have to be solved. Thus, the structure of LI3 is similar to the structure of a linear-implicit method obtained from a diagonally-implicit Runge-Kutta method. Note that the solution of these four systems is computationally cheaper than solving a system of dimension $4 \cdot \dim(\mathbf{x})$ which would result from a linear-implicit method obtained from an implicit Runge-Kutta method. The proof for the stability properties as well as for the order can be found in [14]. Clearly, a time-step with LI3 is computationally more expensive than a time-step with the linear-implicit Euler-method. However, LI3 allows to use larger step-sizes due to the higher order. This is expressed in the engineers rule of thumb that a method of order p should be used for a tolerance of 10^{-p} .

Consequently, stability properties, order and computational complexity make LI3 suitable for real-time simulation of stiff problems and hence hydro-mechanic systems.

Since no iterative algorithm for the detection of zero-crossings can be used, the zero-crossing is assumed to be in the middle of the last solution interval. Note that this leads to an increase in the worst-case runtime of a factor of three.

3 Application Example

In the last section a C++ Simulation-Runtime for the OMC was presented. This simulation-runtime forms



Figure 10: Pieter Schelte (picture taken from [1])

the basis for an automated toolchain for real-time simulation. The workflow of this toolchain is explained in this section using a hydro-mechanical heavy-duty system as an example. In the next subsection the set-up of the real-time simulation is explained. After that some modeling details and simulation results are given.

3.1 Real-Time Simulation Set-Up

The toolchain consists of the OMC as a Modelica Compiler, the C++ Simulation-Runtime, a cross-compiler for ScaleRT and the real-time operating system itself. The hardware setup to execute the real-time simulation of the Modelica model using SCALE-RT requires a host and a target PC. The host PC is standard Windows PC while the target PC uses ScaleRT (Linux with Xenomai real-time extension).

The output of the OMC is coupled to the ScaleRT interface and cross-compiled for ScaleRT. The automation of this step is part of future work. After that the code can be transferred to the target and started via the ScaleRT software in a graphical-user-interface.

Note that in contrast to the OMC neither ScaleRT nor the coupling of the C++ Simulation-Runtime to the real-time interface is Open Source.

3.2 Modeling of the Example System

The application example is a part of a hydro-mechanical heavy-duty system, which is designed to operate on a ship for the installation and removal of oil-platforms. The ship is currently under construction and is pictured in figure 10. The complete hydro-mechanical system consists of eight beams, each with a clamp (or gripper) at the end. During operation the beams move towards the legs of a platform and grip

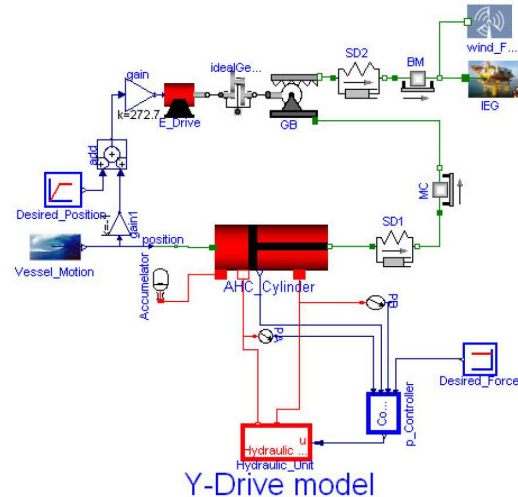


Figure 11: Object diagram of the Y-drive

them. After that the platform can be lifted and removed (details can be found at [1]). Each beam can be divided into a Y- and a Z-drive. In this paper only the Y-drive is modelled and simulated. The Y-drive is used to compensate sea motion, driving the beam towards the leg and applying a constant force towards the leg in case of a contact (in order to avoid hammering). It consists of

- a hydraulic cylinder,
- a 3-way hydraulic valve,
- an electrical drive,
- the beam,
- gears,
- and a force controller.

The electrical drive moves the beam towards the leg using position control, while the cylinder applies a constant force towards the leg during contact using force control.

The Y-drive was modeled in Modelica, where custom models were set up for all Rexroth specific components of the system. Thereby, an incompressible fluid is used. The object diagram is shown in figure 11. Here the hydraulic unit consists of a tank, a pressure source and a three way valve. In order to deal with the large forces inside the system a special kind of cylinder is used and modelled. The flat model consists of 360 equations, while the translated model has 25 state variables and two algebraic loops. The algebraic loops exhibit real as well as discrete variables.

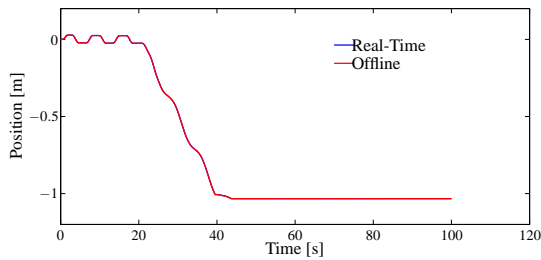


Figure 12: Position of the clamp

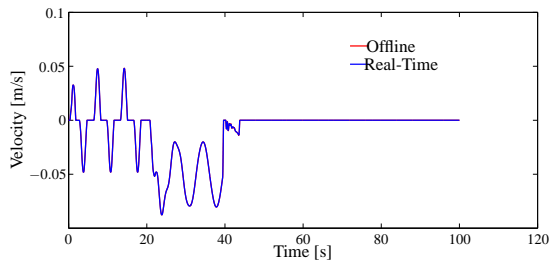


Figure 13: Velocity of the clamp

3.3 Simulation Results

Real-time simulation requires a predictable worst case runtime. Therefore, the number of Newton-iterations in the algebraic loop solver had to be limited. Unfortunately, by doing so it is not guaranteed that an adequate solution is found. Nevertheless, for the used scenario (parameters and inputs) and step size (1ms) a maximum of 4 iterations was required. Hence, the maximum number of iterations was set to 6. The LI3 method described in the previous section was used as numerical integration method. In figure 12 the position of the clamp is shown. The blue line represents the solution computed on the real-time target, while the red line shows the solution computed offline using the C++ Simulation-Runtime and CVode as numerical integration method. It can be seen that the two lines are nearly overlaying. The same holds for the velocity of the clamp shown in figure 13.

4 Conclusion and Outlook

In this contribution the basis for a toolchain for real-time simulation using the OMC is presented. Therefore in section 2 a new C++ Simulation-Runtime was shown that is easy to extend and maintain. Moreover, this Simulation-Runtime includes numerical integration methods, that are suitable for real-time simulation. Due to its flexibility new solvers and algorithms (e.g. multi-rate integration, mixed-mode integration) can be

integrated in the future.

In section 3 the C++ Simulation-Runtime was coupled to the interface of the real-time operating system ScaleRT. That coupling enabled the execution of the C++ Simulation-Runtime together with simulation-code generated by the OMC on a real-time target. The toolchain was demonstrated using a hydro-mechanical heavy duty example system.

In the future this toolchain will be automated, in order to be in the position to generate code for real-time simulation just by a few mouse-clicks. Moreover, coupling of external hardware (e.g. a electronic control unit) is part of future work. This will allow for virtual commissioning using a low-cost toolchain.

5 Acknowledgements

This work is funded by Bosch Rexroth AG and German Federal Ministry of Education and Research (BMBF) in the ITEA2 OPENPROD project.

References

- [1] <http://www.allseas.com/uk/19/equipment/pieter-schelte.html>. website. Accessed: 11/07/2012.
- [2] www.scale-rt.com. website. Accessed: 11/07/2012.
- [3] A. Alexandrescu. *Modern C++ design: generic programming and design patterns applied*. Addison-Wesley Professional, 2001.
- [4] M. Arnold, B. Burgermeister, and A. Eichberger. Linearly implicit time integration methods in real-time applications: Daes and stiff odes. *Multibody System Dynamics*, 17(2):99–117, 2007.
- [5] W. Braun and B. Bachmann. Symbolically derived jacobians using automatic differentiation-enhancement of the openmodelica compiler. *Modelica Conference, Dresden*, 2011.
- [6] F. Casella. Open problems and research trends in oo modelling. Technical report, Politecnico di Milano, Dipartimento di Elettronica e Informazione.
- [7] F.E. Cellier and E. Kofman. *Continuous system simulation*. Springer Verlag, 2006.

- [8] H. Elmqvist, M. Otter, and F.E. Cellier. Inline integration: A new mixed symbolic/numeric approach for solving differential-algebraic equation systems. 1995.
- [9] T. Ersal, H.K. Fathy, D.G. Rideout, L.S. Louca, and J.L. Stein. A review of proper modeling techniques. *Journal of Dynamic Systems, Measurement, and Control*, 130:061008, 2008.
- [10] P. Fritzson, P. Aronsson, A. Pop, H. Lundvall, K. Nystrom, L. Saldamli, D. Broman, and A. Sandholm. Openmodelica-a free open-source environment for system modeling, simulation, and teaching. In *Computer Aided Control System Design, 2006 IEEE International Conference on Control Applications, 2006 IEEE International Symposium on Intelligent Control, 2006 IEEE*, pages 1588–1595. IEEE, 2006.
- [11] E. Hairer, S.P. Nørsett, and G. Wanner. *Solving Ordinary Differential Equations: Stiff and differential-algebraic problems*. Springer Series in Computational Mathematics. Springer-Verlag, 1993.
- [12] A.C. Hindmarsh and P.N. Brown. Sundials: Suite of nonlinear and differential/algebraic equation solvers. *ACM Transactions on Mathematical Software (TOMS)*, 31(3):363–396, 2005.
- [13] L. Mikelsons and T. Brandt. Towards a generic vehicle model. *Journal of Computational and Nonlinear Dynamics*, 7:021013, 2012.
- [14] N. Nikitin. Third-order-accurate semi-implicit runge-kutta scheme for incompressible navier-stokes equations. *International journal for numerical methods in fluids*, 51(2):221–233, 2006.

Time varying mass and inertia in paper winding multibody simulation

Edo Drenth
Modelon AB
Ideon Science Park, Lund, Sweden
edo.drenth@modelon.com

Abstract

This paper will discuss Modelica's unprecedented flexibility for multi-body simulations. Classical multi-body simulation has as a prerequisite constant mass and inertia for deriving the equations of motion for rigid bodies. However, there are industry applications, like the control development of paper winding, that require time dependency of mass and inertia. In these applications mass and inertia cannot be assumed constant and will thus constitute part of the differential equations system by means of introducing mass and inertia as states.

Introducing mass and inertia as states, rather than parameters, requires reformulation of the Newton/Euler formulation of the body model component in the Modelica mechanics multi-body library [3].

A successful new body model formulation has been created and is applied in an industrial example system model.

Keywords: dynamic mass, dynamic inertia, multi-body, mechanics, paper winding, vibration, FMI

Introduction

In the paper industry winding machines are used to reduce the inconveniently large paper roll into smaller paper rolls of just a few tons. The dynamic properties of these machines are heavily influenced by the change in mass and inertia of the paper rolls while winding and unwinding [1, 2]. The time varying resonance

frequencies of the system will put limits on the machines throughput.

The paper industry has an interest to investigate the dynamic machine properties by simulation as the references are proof of. This publication will deal with one of the key aspects of a simulation package to handle; the mass and inertia time (revolution) dependency.

Many specialised multi-body packages are built upon constant mass and inertia's to solve the equations of motion. To cope the problem of varying mass, the system is analysed at different points of operation rather than simulating a full run.

Mastering this topic of dynamic mass and inertia properties may not only allow for system controllers' validation in the time domain with Dymola's¹ real time capabilities, but also support algorithm development with FMI [6] technology exporting models to control development environments.

This publication shows Modelica's capabilities in to this specific topic of paper winding.

Modelica Body

Mass rate signals

In order to create sound models, which can be diagnosed upon dimension consistency, mass rate signals are defined. These read,

```
type MassRate =  
  Real (final quantity="MassRate", final unit="kg/s");
```

¹ Dymola is a registered trademark of Dassault Systèmes

```
type MomentOfInertiaRate =
  Real (final quantity="MomentOfInertiaRate", final unit="
  kg.m2/s");
```

Newton/Euler equations

The Modelica Standard Library defines bodies with the help of Newton/Euler equations around the centre of mass of the modelled body. These equations have to be elaborated to count for the mass and inertia rates. From Newton’s second law we have

$$p = mv$$

$$F_{net} = \frac{dp}{dt} = m \frac{dv}{dt} + v \frac{dm}{dt}$$

In the above equation F_{net} is the net external force applied to the system, since Newton’s second law is only valid for constant mass particles [3]. Reference [4] exemplifies how a net force can be derived, hence

$$F_{ext} + m_w \frac{du}{dt} + u \frac{dm}{dt} = m \frac{dv}{dt} + v \frac{dm}{dt}$$

The net force on the left hand side of the above equation includes a so called “thrust” force from the mass flow, $\frac{dm}{dt}$. In our application a difficulty arises to determine the web mass, m_w , which is accelerated. At constant web velocity, u , this term vanishes.

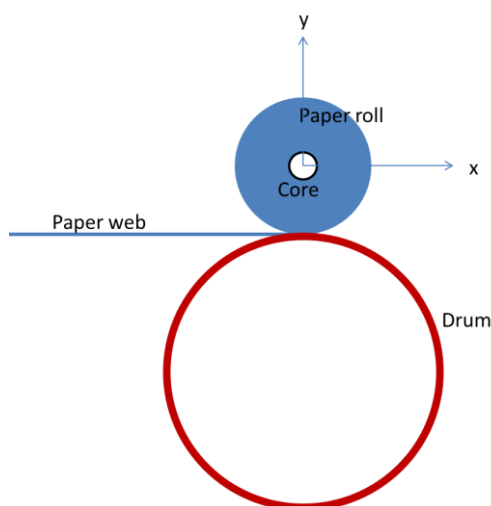


Figure 1 Sketch of model

With help of the right hand side of the force equation, the Modelica code for a body in 3D will be modified and yields,

```
frame_a.f = m*(Frames.resolve2(frame_a.R, a_0 - g_0) +
  cross(z_a, r_CM) +
  cross(w_a, cross(w_a, r_CM))) +
  mdot*(v_0 + cross(w_a, r_CM));

frame_a.t = Idot*w_a + I*z_a +
  cross(w_a, I*w_a) +
  cross(r_CM, frame_a.f);
```

In the above set of equations $mdot$ will be defined as mass rate and $Idot$ as a 3x3 inertia tensor with pivots of moment of inertia rate signals. The above mentioned “thrust” force from reference [4] will have to be applied externally to $frame_a$ as forces and moments.

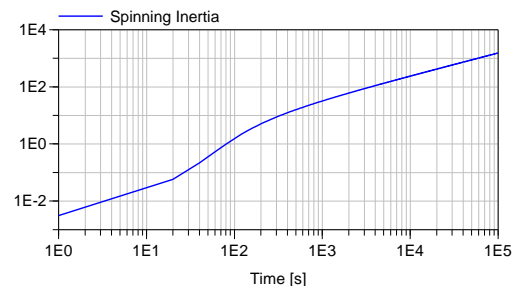
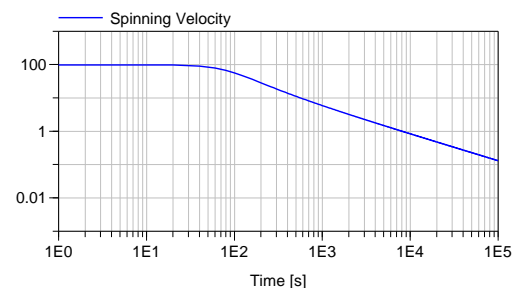


Figure 2 Free spinning results with constant impulse momentum

Body properties

A special variable mass and inertia model has been designed that can resemble the effects of increasing radius of the paper winding roll. The radius (rate) of this body defines both mass (rate) and moment of inertia (rate) too. The inertia tensor however, and especially the rate, is application dependent and a generic solution difficult. Hence, the paper deals with paper winding only. All properties are dynamically sound for a multi-body analysis.

E.g. a free winding roll with initial velocity will reduce angular velocity over time due to increased spinning moment of inertia (see Figure 2) at constant momentum (net external force equals zero).

Difficulties arose by straight forward modelling of variable mass and inertias. The symbolic manipulation of the equations showed imbalance on the number of equations and unknowns. These are overcome with simple and physical sound, mathematical reformulation of the existing body model in the Modelica Standard Library.

This dynamic mass and inertia body model is accessible from a library and has become a building block for usage in other models for mechanical simulation.

Model Assumptions

The model envisioned shall deal with the simulation problem of variable mass and inertia solely. One drum, with the drive input, and one paper roll (blue coloured in the animation series below) are modelled. The interaction between the drum and paper roll is based upon a simple impact force function for the support forces and a simple load dependent slip force with relaxation for the horizontal forces (see Figure 3). These horizontal forces will apply a drive torque on the paper roll and a torque load on the driving drum.

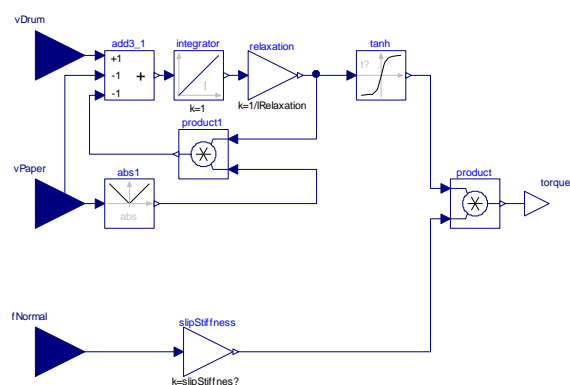


Figure 3 Load dependent slip force with relaxation

No special emphasis is made on material properties of paper with respect to friction and material damping, nor the web tension's

influence on the dynamic performance. The models are purely made to show Modelica's capabilities of the specific problem of variable mass and inertia.

The paper roll can move freely in the vertical direction and around its spin axis only, for reasons of simplicity and the limited scope of this report the other four degrees of freedom are kinematic constraint.

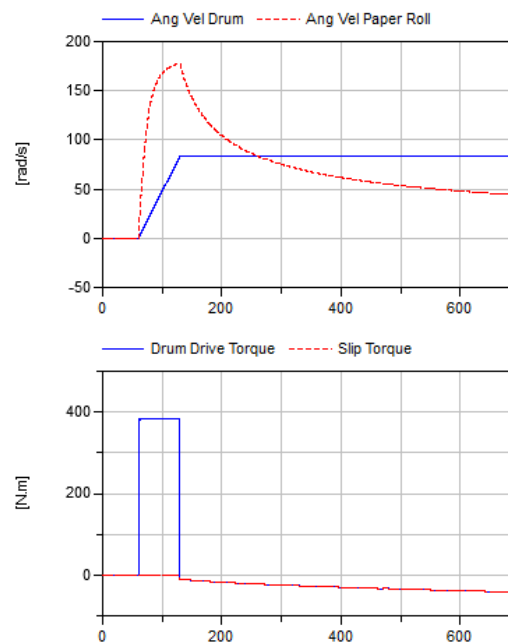


Figure 4 Speed and Torque profile of Drum and Paper Roll

Load case

The drum is driven with a speed profile as indicated in Figure 4. The speed profile equates to certain constant web acceleration until a predefined velocity is reached. This velocity is kept constant throughout the remaining simulation. For the sake of comparison the direction of operation is taken positive, whereas in the real application the drum and paper roll rotate in opposite directions.

The centre of mass of the paper roll has a small offset from the centre line in order to introduce vibrations in the two drum system. These induced vibrations are solely for the sake of exemplifying the time varying oscillations.

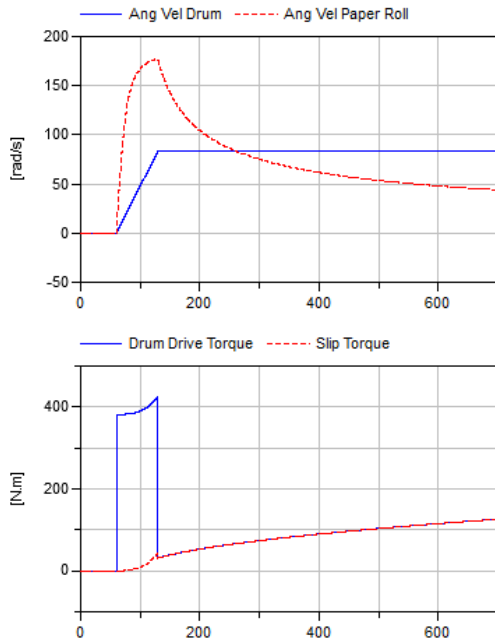


Figure 5 Speed and Torque profile of Drum and Paper Roll

Discussion of Results

Speed and Load

The drum is driven with a speed profile as indicated in Figure 4, the paper roll angular velocity is determined through acceleration of the same by means of a frictional force at the interfacing surface of roller and drum. Because the radius of the paper roll starts at the core radius, which is much smaller than the drum's radius, and increases over time the angular velocity increases steeper than the drum's angular velocity. At approximately 260 s (with the used data) both radii are equal and hence the angular velocity is equal too.

The speed profile of the drum will require a driving torque as indicated in Figure 4 above. The initial large value is due to the acceleration of the heavy drum. Once the desired velocity of the web is reached the torque level is determined by the increase of the paper roll spinning inertia. This paper roll inertia increment becomes a torque load (red in Figure 4) upon the system. Surprisingly the actual load is negative, because the incoming paper from the web results in a system accelerating torque from the linear momentum (web tension

omitted), because its forces are applied tangential at the paper roll surface. To keep a constant web velocity the driving drum torque is negative.

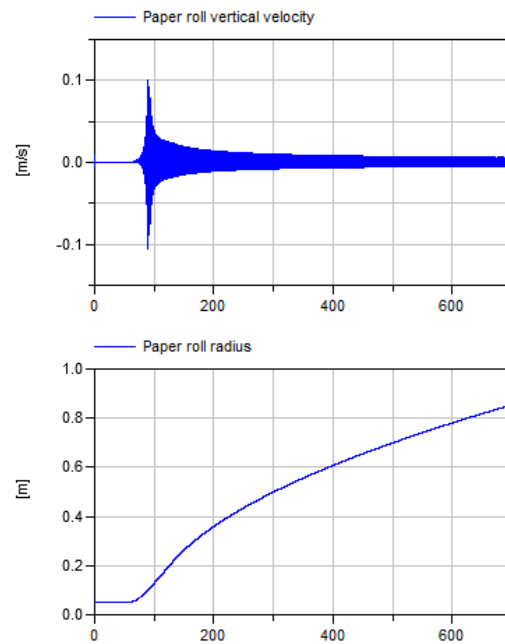


Figure 6 Roller vertical velocity and radius over time

Figure 5 depicts results when the web linear momentum is omitted. The increasing inertia yields an increasing torque.

The offset on the centre of gravity results in a forced vibration of the paper roll in the vertical direction (Figure 6) and around its spin axis (Figure 7). The paper roll is constraint in the remaining directions, but could technically have compliant bearings. This is outside the scope of this paper.

The radius of the paper roll increases over time and the increasing radius will also create vertical motion of the centre of gravity. The mean value of the vertical velocity is positive indicating the centre of gravity rises.

Radius and velocity

The radius rate of change decreases over time, because the web velocity is constant and the actual radius increases. The paper roll's tangential velocity is constant, hence the angular velocity must decrease and thus the radius rate.

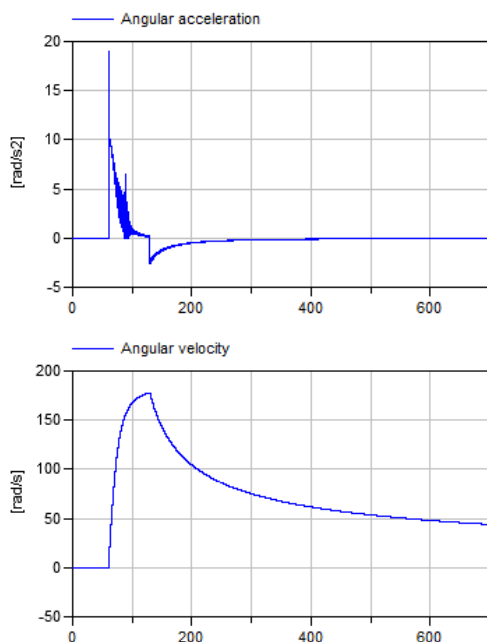


Figure 7 Roller angular velocity and acceleration

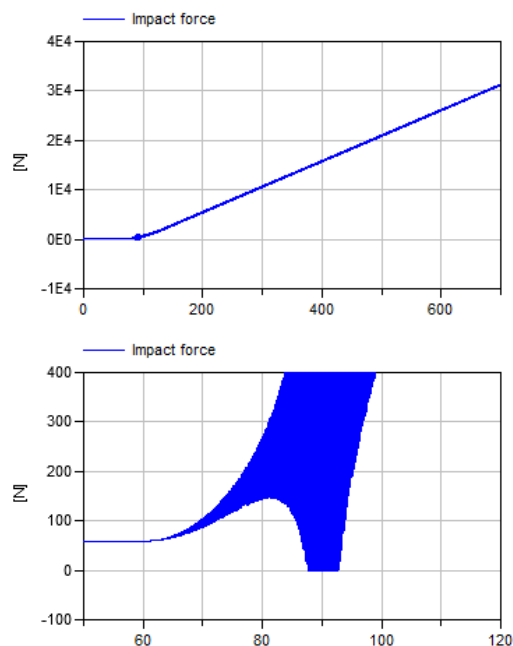


Figure 8 Impact force between drum and paper roll

Impact Force

The paper roll is supported with help of an impact force between the drum and the paper roll. Due to the fact the mass increases over time the impact force will increase too (see Figure 8). A close up is made to show the lift of at around 90 s of simulation time.

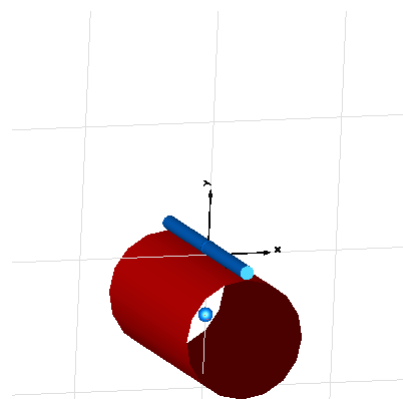
Do mind that the impact function, with viscous damping only, may not at all be representative for paper properties. Modelica is however, very well suited to accommodate any impact model and thus able to model material damping, but again outside the scope of this paper.

Bending and torsion

The discussed and simulated model has neither bending compliance nor torsional compliance. These effects are omitted in this paper, but will have an impact on the dynamic behaviour of the real system. One could however, make a roller and a drum component that can be compliant connected to another roller and drum component easily in Modelica. This way a discretized lumped mass and inertia roller and drum system is created to reflect the bending and torsional vibrations of the system.

Animation

Dymola has capabilities of dynamic graphical presentation of the variable mass and inertia body. Below an animation sequence with a sample every 100 s is presented of the simulation run discussed in this paper.

Figure 9 Animation, $t=0$ s

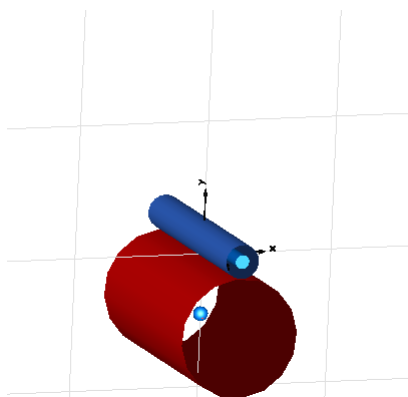


Figure 10 Animation, t=100 s

The animation clearly depicts the increase in radius and the centre of gravity rising over time.

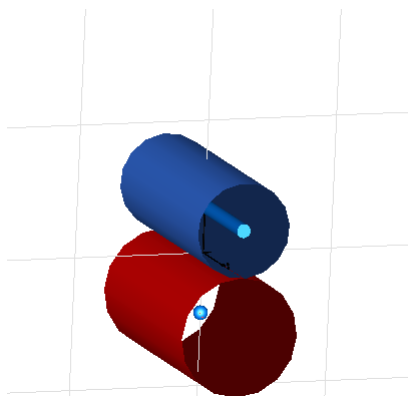


Figure 11 Animation, t=200 s

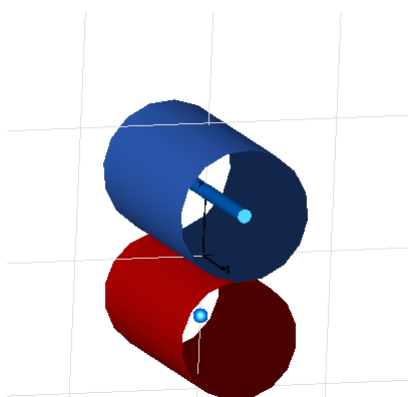


Figure 12 Animation, t=300 s

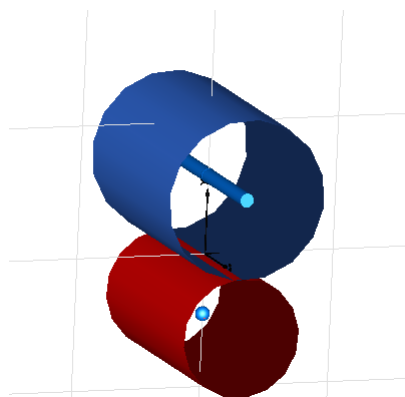


Figure 13 Animation, t=400 s

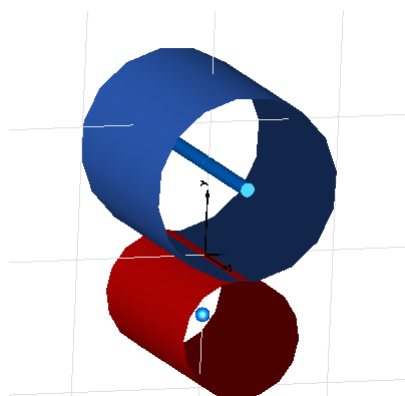


Figure 14 Animation, t=500 s

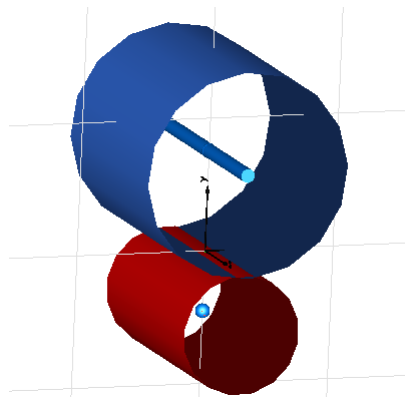


Figure 15 Animation, t=600 s

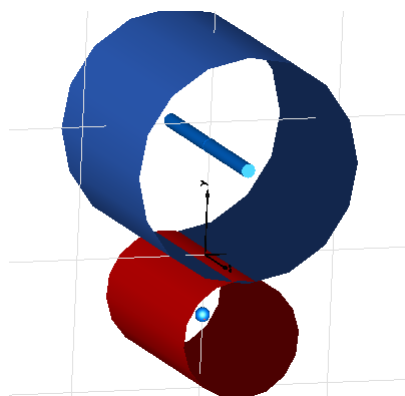


Figure 16 Animation, $t=700$ s

Functional Mock-up Interface

The newly developed dynamic mass and inertia models can also be used for controller development in specialised environments like Simulink². The FMI Toolbox from Modelon is used to import the Functional Mock-up Unit exported from Dymola. The result is depicted in Figure 17.

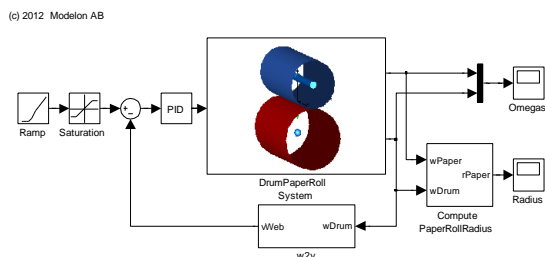


Figure 17 Simulink model with FMI block of the Drum-Paper Roll mechanism

This model exchange allows the control engineer to have an excellent non-linear plant model to develop his algorithms against.

Dymola's real time capabilities allow the user to export the models to a HIL system and verify the actual electronic controller.

Conclusions

This report shows that Modelica is capable of modelling variable mass and inertias of winding machinery. A special body element is created in the Modelica language which becomes a reusable body object. This body can be used to model a lumped system of a roller to accommodate bending and torsional modes (future work).

The created dynamic mass and inertia model is a prerequisite allowing for virtual controller software development, verification and validation at a systems level for a complete winding cycle.

References

1. Virtanen T, *Fault Diagnostics and Vibration Control of Paper Winders*, Ph D dissertation Helsinki University of Technology, Espoo, Finland, 2006
2. Zwart, J., Tarnowski, W., *Winder Vibration Related to Set Throw-outs*, PAPTAC 89th Annual Meeting, Montreal, Canada, 2003
3. Kleppner, D, Kolenkow, R., *An Introduction to Mechanics*, McGraw-Hill, pp. 133–134, 1973
4. Chandler, D., *Newton's Second Law for Systems with Variable Mass*, The Physics Teacher, Vol. 38, 2000
5. Modelica Standard Library, Version 3.2
6. 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*, 8th Modelica Conference, Dresden, Germany, 2011
7. *FMI Toolbox User's Guide 1.3.1*, Modelon AB, Lund, Sweden, 2012

² Simulink is a registered trademark of The Mathworks

Collaborative complex system design applied to an aircraft system

Eric Thomas, Michel Ravachol

Dassault-Aviation

78 quai Marcel Dassault Cedex 300, 92552 St CLOUD Cedex, FRANCE

eric.thomas@dassault-aviation.com michel.ravachol@dassault-aviation.com

Jean Baptiste Quincy, Martin Malmheden

Dassault Systèmes

10 rue Marcel Dassault, 78946 VELIZY-VILLACOUBLAY Cedex, FRANCE

jean-baptiste.quincy@3ds.com martin.malmheden@3ds.com

Abstract

Aircraft systems have evolved dramatically since the beginning of aviation. Many improvements of performance and safety have been made. Now each sub-system has optimized performance and it is thus difficult to find gains without breakthroughs in architectures or technologies; and this is the objective of the R & D studies towards a more electric aircraft.

Simulations are widely used to explore and justify aircraft architectures [1], but system simulations currently suffer from limitations which make them difficult to use for complex multi-systems analysis. Therefore tools and processes must evolve to accompany these major changes in order to support the designers in their quest of optimized design.

This article deals with new processes and tools which will take part, in a close future, in the determination, the verification and validation of systems architectures. The results presented here were obtained during the CSDL project (Complex Systems Design Lab) partly funded by the French government.

Keywords: Collaborative process, System engineering, MBSE, hybrid DAE, multi-physics, multi-levels, Optimization, Robust Design, Coupling Simulation System - surrogate Models, PLM/SLM integration

1 Introduction

Aircraft vehicle systems are typical examples of complex systems. They are composed of many sub-systems, which overall represent a set of thousands of equipment, and that have more and more interactions between them.

These sub-systems are provided by several companies for integration and must fulfill aircraft requirements.

The efficient study of performance and safety is of prime interest when designing complex systems in a collaborative context. At each stage of the design cycle, system engineers should be able to find optimized architectures of systems according to requirements. Such need is particularly important in the early stages of design when decisions on the aircrafts concepts, systems architecture and partners choice will determine the performance and the future cost of the product.

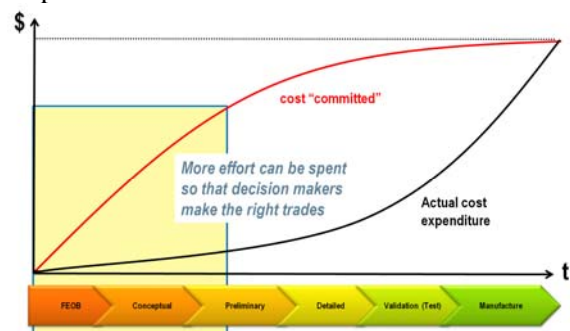


Figure 1 Design phases and effort ramp-up

It is thus necessary to make the right decisions in these early phases. With this intention, the systems architects may find it beneficial to explore spaces of design in a smart automated way in order to identify the points of interest quickly.

The article is structured as follows:

- Section 2 briefly presents aircraft vehicle systems and their design process.
- Section 3 details the requirements for a collaborative tool for complex system design.

- Section 4 explains the solutions developed within the project CSDL, in particular collaborative management of hierarchical multi-physics Modelica models with Dassault Systèmes V6 PLM platform.
- Section 5 presents the challenges ahead to get a full and efficient set of tools and processes for future airplane designs.

2 General information on Aircraft vehicle systems

2.1 Architecture

Aircraft vehicle systems are composed of several sub-systems. The main sub-systems are represented in the following composition (figure 2), here for a conventional aircraft architecture. There are Environment Control System (ECS), Power Plant (engines), Electrical, Braking, Hydraulics or Fuel systems.

They are connected together and to several other parts like control systems, passengers, environmental conditions or system properties [2].

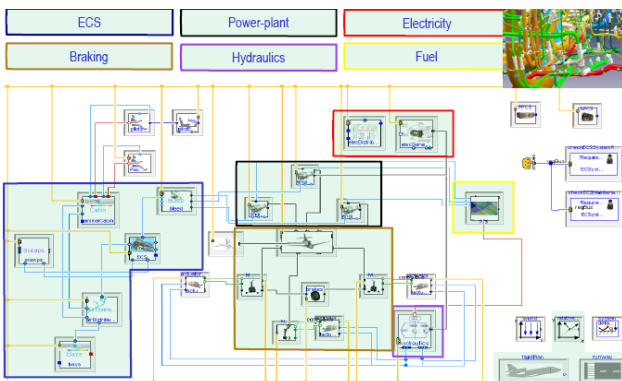


Figure 2: Systems architecture of a conventional aircraft

Each of these sub-systems is itself composed of sub-systems or equipment. For example the ECS is composed of a bleed (which mixes air flows from the engines), a cold air unit (which in particular manages cold and hot air flows to achieve a good comfort for the passengers in the cabin and a sufficient cooling for the equipment in the bays), a distribution sub-system (pipes and parts for distribution of air to cabin and bays), scoops (to get cold air from external environment ...).

The ECS is connected to engines (power-plants), passengers comfort models, and environmental conditions.

Traditionally, each aircraft system is defined within ATA (Air Transport Association) numbering,

which provides a common referencing standard for all commercial aircraft documentation e. g Chapter 24 is for electrical power or 21 for Air Conditioning and Pressurization. This standard has many benefits on common decomposition of aircraft functions, but tends to segregate sub-systems that may be optimized nearly independently from each other.

In this conventional architecture (fig. 2), electrical systems have only limited interactions with other subsystems. In the case of a more electrical aircraft, that is one of the most significant technology changes for the near future with many expected benefits, electrical equipment will be spread across multiple systems. Therefore, there is a consensus that the way to the truly optimized complex system is through an overall system redesign, including a trans-ATA approach.

2.2 Aircraft systems main activities and design process

The main activities of people involved in the aircraft vehicle systems are:

- System design and integration of the vehicle systems on the aircrafts.
- Follow-up, technical expertise and technical facts processing for the aircraft in service.

The activities are then not limited to design systems only, but also participates in the maintenance, improvements of the aircraft systems along the whole life-cycle (more than 30 years) and in decommissioning.

These activities include participation in the certification process of the aircraft which is necessary to allow the plane to fly; and for which product justification and traceability with respect to the requirements are mandatory.

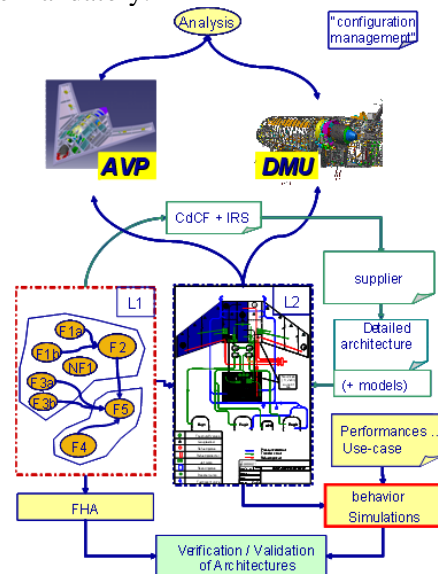


Figure 3: Several of aircraft vehicle systems activities

In the image above a typical workflow between the aircraft manufacturer and its suppliers and partners is presented. Several activities of the verification and validation process made during functional analysis, analysis which participates in justifications (e.g. FHA - Failure Hazard Analysis, behavioural analysis) are also sketched, connected to functional, logical and physical architectures.

3 Requirements for an aircraft systems design platform

Now, it is possible to list requirements for a truly efficient collaborative platform for aircraft systems design and optimization.

3.1 General requirements

The “must have” features of such a collaborative design tool can be listed as follows. They must allow:

- Compatibility with the tool managing the definition: currently 3D Digital Mock-up with Product Lifecycle Management.
- Project management during the entire lifetime of an aircraft (more than 30 years)
- Collaborative work between all stakeholders of the design of the aircraft systems.
- System engineering process: requirements, functional and architecture management e.g. standard architecture descriptions according to ATA decomposition.
- Several architecture analyses, in particular behavioral simulations, based on 3D and system representations.

3.2 Requirements on models for aircraft systems architecture

Model Based System Engineering (MBSE) is a key practice to advance complex systems development and Modelica is a critical enabler of MBSE

But system architecture analysis based on models must also be architecture driven because it is the architecture which must be justified and optimized. Simulations are means for architecture assessment. Therefore, it must be possible to add behaviors to components of sub-systems or directly to the sub-systems of an architecture.

Tools and models must also have several features as described bellow.

3.2.1 Tools for performance analysis

To evaluate the performances of systems architectures during trade-off, analysis based on simulations are widely used, from simulation of 0-D/1-D models to multidimensional models (FEA/CFD...) for more detailed analysis. The architecture components should thus be able to have models with multiple levels of details, chosen according to needs.

In fact, designers should have all models needed to model the behaviors they want according to the types of analysis that are to be done. Current analyses are listed bellow:

- Static analysis for study of energy balance, energy flow distribution or of particular design points
- Dynamic analysis to study analysis along time, or Eigen values.
- DOE (Design Of Experiments) including sensitivity, robustness and optimizations analysis.

And this must be applied on models with nominal and non-nominal behaviours (e.g. when failures occur).

There are also requirements on simulations management, because simulation properties, models, results must also be stored and managed to be usable many years later.

3.2.2 Libraries of models for system engineering

What kind of features would a system engineer like to find in the application libraries?

System engineer wants to have a set of models able to represent the behavior of physical components with a sufficient accuracy for the kind of analysis he/she has to do, and to focus on technical subjects in the way to chose and optimize systems. Then, system engineer would like to find:

- Multi-domain and multi-physics libraries of components for the large range of physics implied in the aircraft systems.
- Versatile components whose physical properties can be parameterized according to product data sheets or with data linked to definition (managed by the PLM.)
- Application libraries with validated components should be valuable, if not essential. Validation in a defined range of application is very important, because it is the base for the re-use and extends of components (which contains knowledge of the company).
- Switches to enable a model validity checker or not. Supposing that a validity model is already defined (see properties [2]).

- Switches to define physical hypotheses: consider static or dynamic behavior, nominal or non-nominal behaviors.
- Published additional data which can help to set simulations. For example stochastic data are often added to models afterwards by system engineers. It is not logical that such information is not usually included in models provided by partners. In fact they are the best specialists for publishing such useful information at the right places in the models. A general mechanism for publishing such data should be studied to enable this process.

3.3 Requirements for model interfaces and model exchanges

To allow connectivity of models (equipment or sub-systems), it is important that standard interfaces are defined, and that more complex interfaces could be derived from them. These standards must be applied by all partners, and managed like other interfaces.

The tool shall manage:

- IP for model exchange (integrate models of partners, provide to partners system models).
- Interface between sub-systems. In particular it must allow change of components (sharing a particular interface) as defined below in the application example when surrogate models may change.

After decomposition in black or grey boxes, simulations of systems should remain efficient (see requirements below.)

Functional Mock-up Interface, FMI [7].) can be used for encapsulation of Modelica models and other model code as soon as it respects previous requirements.

3.4 Requirements for simulations

For early verification of an architecture, quick evaluations based on thousands of simulations are required to explore the design space. Therefore, system simulation is often used because it is far quicker than 2D-3D FEA or CFD computations. They are used to find robust and optimized designs by use of sensitivity, robustness and optimization process. They must also take into account variability of architectures, parameters defined as a range or as a stochastic distribution. It is also necessary to be able to increase granularity of certain equipment models that has proven particularly sensitive or to incorporate new observers only available in detailed models.

Tools often allow co-simulation between 0-1D models and 2D-3D models. However, it is not really adapted to early verification because they can lead to slow simulations, which are often not compatible with efficient optimizations processes (with several parameters to optimize and having multiple criteria) which require a large number of computations.

Computation time is critical because simulations must be feasible within time constraints to get results, analyze them and choose the optimized architecture with a good level of confidence often after several interactions. It is also important to have sufficiently fast simulations in order to make early decisions and explore alternative architecture designs during a decision review. To allow such quick calculations High Performance Computing (HPC) features, parallel computations, and distribution of simulations on adequate hardware are other key factors.

3.5 Requirements for model debugging

The previous sections suppose that models simulate without problems. But it is well known that complex systems written in a natural physical language such as Modelica often gives sets of hybrid Differential Algebraic Equations with non-linear equations that can be difficult to initialize and solve.

Even if Dymola and DBM, the Dymola kernel integrated in Catia V6, is very efficient; performance and convergence of the initial problem also depends a lot on the quality of the code written by the author of the model as well as the how well the iteration variables of the initial problem have been/can be set. Features like the homotopy operator [5] help the user to solve initialization equation systems by providing a simplified model requiring less start values of iteration variables of the initial problem. However, it is important that such features could be used both by model developers and by final advanced users (see published properties and features in next section).

It is also important that the simulation tool help users to localize the cause of problem. Many features have been introduced in Dymola. Following new features can help:

- More (visual) features to quickly locate important information (e.g. component highlights, model comparisons ...).
- Structural analysis to study architecture of models to localized ways of simulation improvements (causality, algebraic loops, invertibility ...).
- Other methods will be studied in the near future [8], in particular Modelica models with structural changes and non-nominal behavior integration.

4 Application to an aircraft system

Investigation of these problems for very early stages of design have been done within the project CSDL, which had the objective to develop a comprehensive collaborative environment for decision making at the earliest stage of a project.

It tries also to take into account that process and associated tools must help designers along all the lifecycle of an aircraft, from early stage to operational service, including justification to requirements traceability.

It is applied to the design of an environmental control system.

4.1 Description of the system

An environmental control system (ECS) was selected because it combines several demonstrative features which can be applied to other systems afterwards.

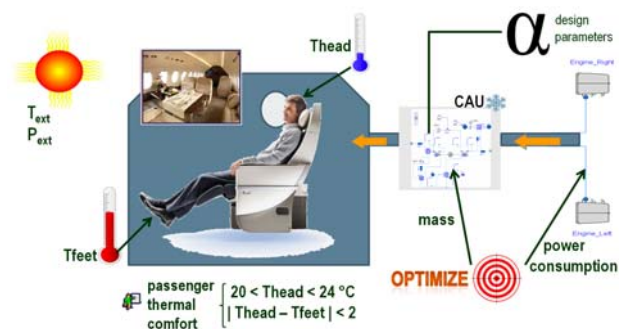


Figure 4: ECS Sizing engineering problem

For this reason, a generic model of ECS was previously used as a base during ITEA2 Eurosyslib [6] for properties modelling (see [2]) and will be used to enhance several modelling features during ITEA2 Modrio [8]. In CSDL it is used to investigate multi-level modelling and collaborative design.

This generic model is a 0-1D model written in Modelica. It is composed of basic sub-systems. Air flow comes from two engines modelled as boundaries with fixed pressure and temperature. A bleed mixes the two flows and provides the resulting flow to the Cold Air Unit (CAU) which regulates mass flow and energy given to the Cabin. Usually the energy flow rate coming from the CAU is provided to the different parts of the Cabin and to the Bays through a complex piping system. In this example, only a Cabin is taken into account.

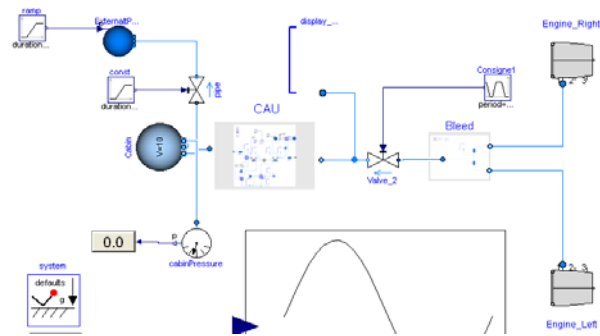


Figure 5: Generic ECS

The CAU is composed of a compressor, a turbine, heat exchangers, pipes and a regulating valve controlled by a PI controller which uses the measured Cabin temperature and a temperature set point for the regulation, as shown in figure 6.

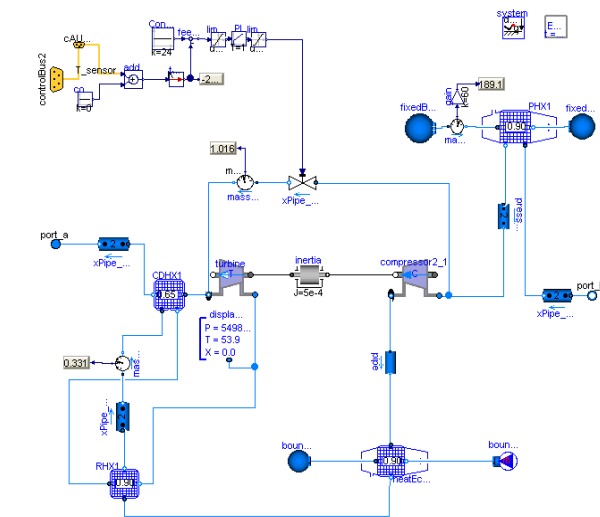


Figure 6: Cold Air Unit

4.2 Surrogate model

For rough assessment, a cabin modelled as a volume or some combination of volumes and heat wall exchanges may be sufficient. But, for more detailed insight, in particular for passenger comfort, it is more suitable to calculate the air flow in cabin using CFD codes. A usual method is to co-simulate the two models. For assessment based on small number of calculations, it is possible to do this; but optimizing the system may require too many simulations to be run.

As for passenger comfort optimization, where insight of only a couple of variables in the cabin are required, it is better to build a reduced surrogate model from CFD and optimise the system using it as described in figure 7.

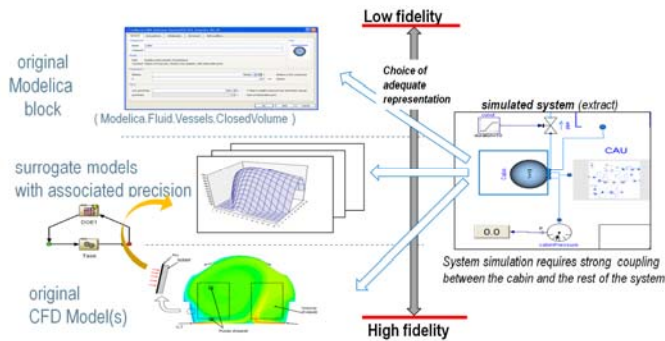


Figure 7: Cabin modeling options

Several types of surrogate models can be used to approximate the CFD response, RBF (Radial Basis Function) being one of them. A surrogate model is a parameterized function. In our use case, inputs are temperature T and velocity u of the air injected into the Cabin, plus external temperature T_{ex} . Output are temperatures at several selected points in the cabin: T_{feet} , T_{head} and T_{sensor} which are temperatures around passenger feet and head, sensor used for temperature control feedback.

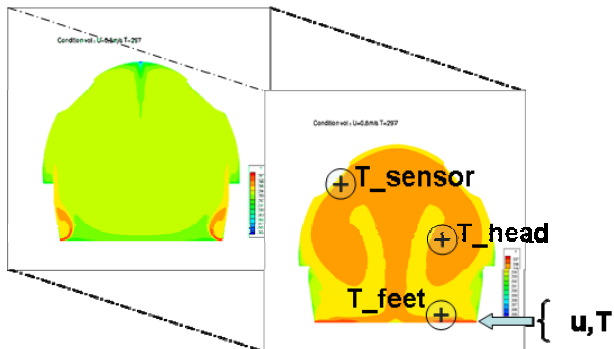


Figure 8: Surrogate model inputs and outputs

The function is expressed by a mathematical formulation that is parameterized by a set of weights. These weights are computed so that the surrogate model matches the CFD response.

4.3 Surrogate model integration in Modelica

To integrate the new model, we need to modify the interface between the CAU and the Cabin to define a common interface that is usable for a number of models both Modelica native and imported ones.

Exchangeable models are declared as replaceable and constrained by the specified base model which manages the interface connections compatibility to other sub-systems. It is done in a similar way as made in the Modelica library called VehicleInterface.

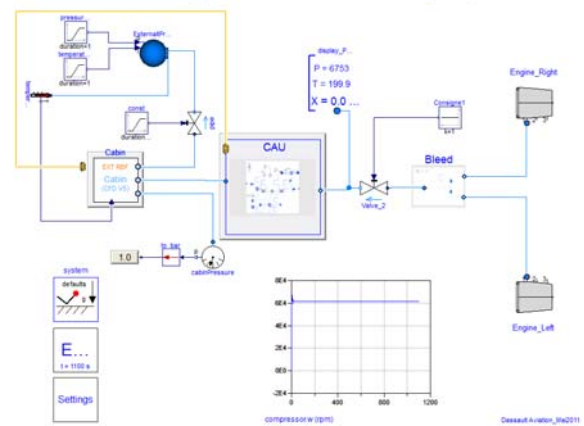


Figure 9: Modelica model with replaceable components

Therefore, when implementing the system template with a new Cabin representation, only models having a compatible interface are proposed to the user, as shown in the following figure.

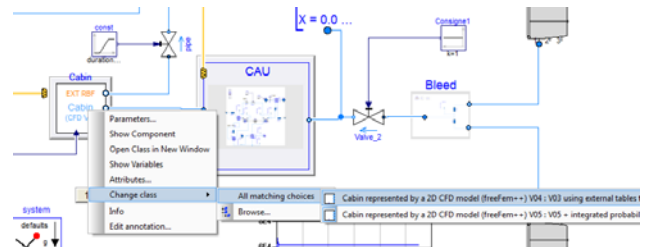


Figure 10 Interchangeable Cabin models

Internal parts of compatible models are then defined as can be seen below:

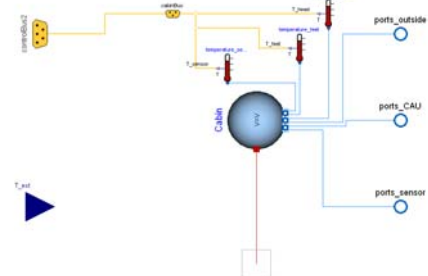


Figure 11: compatible interface with a Volume model

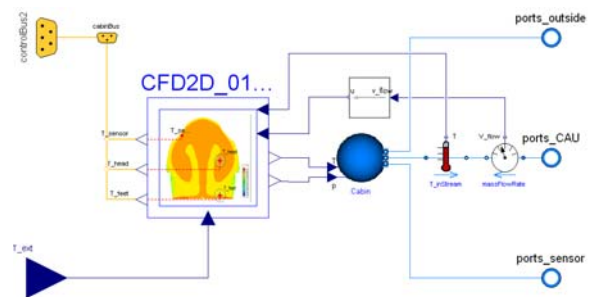


Figure 12: same compatible interface with a CFD model

Models allowed to be used can be both native Modelica models, reduced models such as RBFs implemented in Modelica but reading data exported from Isight at runtime, FMU's or other. This ap-

proach shows how a flexible common system structure can be defined using the redeclare/replaceable constructs to allow simple configuration of a large number of architecture design alternatives incorporating different levels of granularity and origin of the subsystems depending on what is the subject matter.

4.4 Stochastic distribution in Modelica

Stochastic properties of parameters used for analysis like robustness are often added to model afterwards when needed. Such properties should be associated to the model by the company that provides products or sub-systems; Attempts to define standard definition of uncertainties have been done, e.g. as expressed in [4]. But it is not yet standardized, even if it should be.

We have then tried to add these properties in a way that will be easy to use for adding such metadata in existing models. It is done by defining base classes for distributions and extending the models with these base classes (here adding a tab in the Dymola graphical interface with additional parameters for probability distributions.)

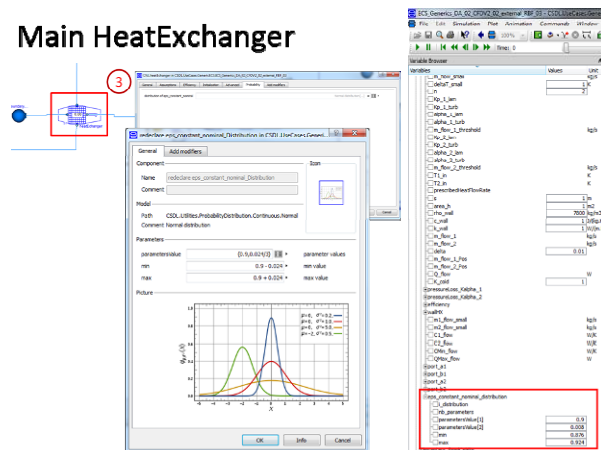


Figure 13 Stochastic data definition within Dymola

Such meta-data should be managed by the tool with publishing mechanisms. The following figure shows an Isight workflow where these stochastic properties defined within the model are mapped in order to be reused in a robustness analysis.

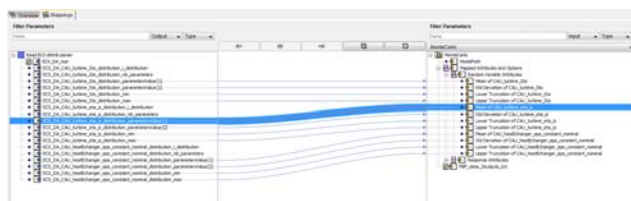


Figure 14 Stochastic data extraction

4.5 Design process

During the design process, several activities must be carried out. Only main ones are presented. The purpose is not to be exhaustive, but to show workflows and illustrate what must be done and how it could be done. These activities are iterative and must create formal links between architectures and products with valid requirements (see [1]). They are also collaborative (see next chapter).

4.5.1 Engineering Requirements

Passenger thermal comfort should be guaranteed for a whole range of operating conditions. Some specific operating conditions corresponding to external temperature extrema have been chosen as dimensioning test cases.

Moreover, several objectives have been set: minimum mass for the system, minimum mass flow rate extract from the engines.

Among all design space parameters of the model, several parameters have been selected: turbine efficiency and nozzle area, main heat exchanger efficiency

More types of requirement for an aircraft ECS may be found in [2].

4.5.2 Functional analysis and Logical architecture

A simple decomposition of the functional and logical views are presented in next figure. The functional view represents what the system should do, and the logical view represents how it is implemented. The logical view shows here that Engines and ECS are parts of two different ATA (ATA 71 for "Power Plant" and ATA 21 for ECS, exactly "Air Conditioning and Pressurization")

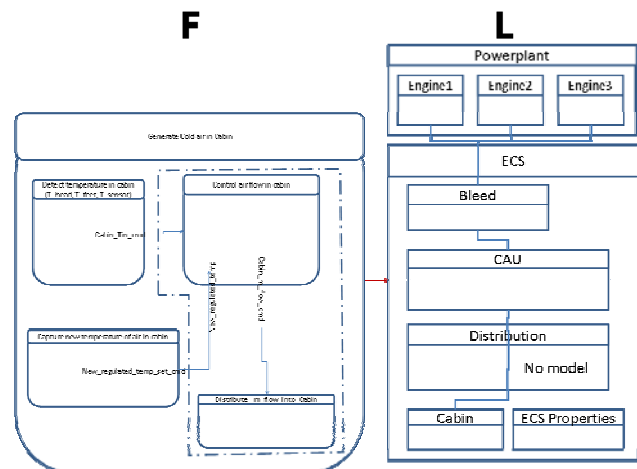


Figure 15 Functional and Logical Views

4.6 Collaborative process

To manage systems and build previous models, specific skills are required. Several actors may interact in aircraft manufacturer units or in partner companies.

4.6.1 Actors

To study the collaborative process, several actors have been identified and defined in the following table:

Actor		Role description
Project Lead		Responsible of the planning the engineering activities within the project
Architect Aircraft		- In charge of aircraft architecture - Specify targets for subsystem and arbitrates trade-offs between subsystems.
Thermal Architect		- In charge of architecture for the air-conditioning ECS subsystem - Studies alternatives and optimizes the design
CAE (CFD) Analyst		In charge of CAE based studies in his field: thermal studies - CFD multi-D calculations - Surrogate models generation
Method Engineer(s)		Provides a set of prepackaged simulation services that can be used by CAE analysts, system architects and specialists

Figure 16: Set of involved actors

4.6.2 Collaborative Workflow

A workflow describing the engineering process has been defined, as shown below.

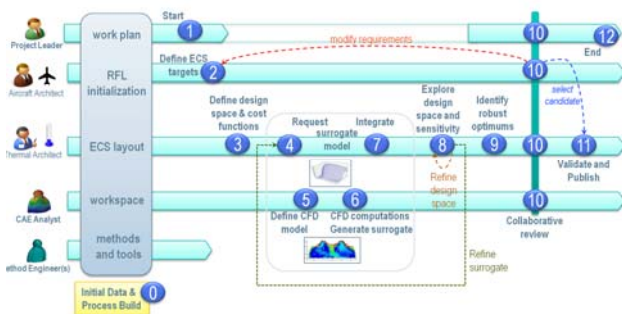


Figure 17: Collaborative process

Some of the steps are supported by a simulation services automated in Isight, as described below.

4.6.3 Design process

4.6.3.1 Sensitivity analysis

Sensitivity is the first analysis performed on a model. It helps identifying important parameters to focus on, and parameters on which tolerances may be relaxed.

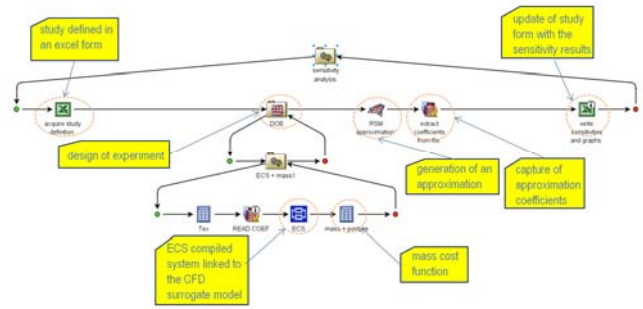


Figure 18 Isight sensitivity analysis configured by a spreadsheet

4.6.3.2 Optimization

The final aim is to produce optimized systems according to multi-objective requirements. It is then an important activity among all design activities.

4.7 Leveraging V6 RFLP

As we mentioned earlier, efficient collaboration between stakeholders is a key ingredient. V6 CATIA Systems enables such collaboration by:

- Providing a unique data referential for requirements (R), functional decomposition (F), logical product definition (L) including 0-1D models (cf. lower part of picture 20 showing the ECS), physical product definition (P) including CAE multi-dimensional models.
- Tracing dependencies of these data through implement relationships (cf. right hand side of picture 20 showing implemented/implementing relationships thru the R-F-L-P cascade),
- Tracing additional dependencies by capturing data flow of simulation processes (detailed in next section).

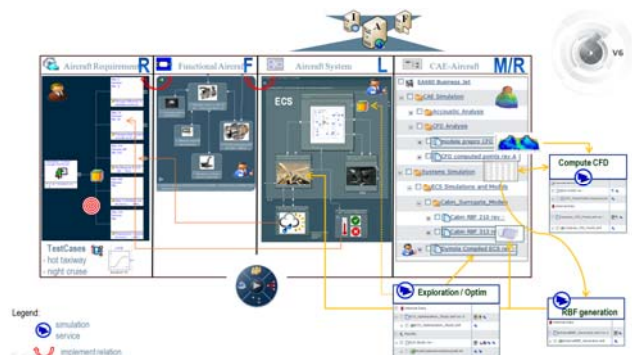


Figure 19: Mapping of use case data to V6 data referential (RFLP and Simulation)

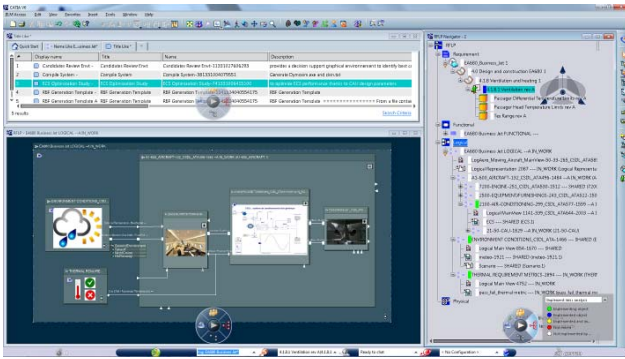


Figure 20 Thermal Architect V6 cockpit: ECS RFLP (right), ECS system (bottom) and design exploration services (top)

In this way, not only each stakeholder can manage the lifecycle of his/her own data properly but also have access to the data published by other stakeholders and author his/her data in this context. Out-of-sync situations can be properly detected in the case some upstream data is revised.

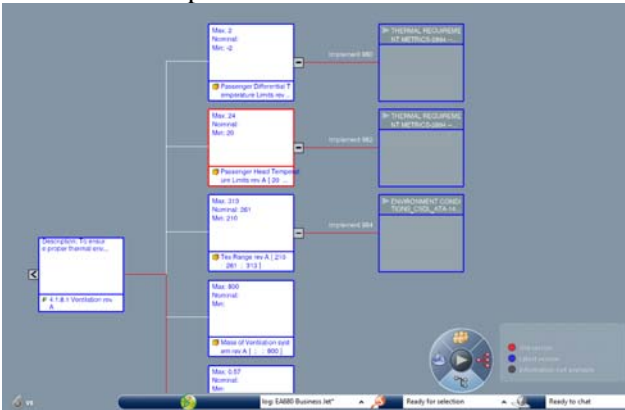


Figure 21 Compass showing that a system reuses an old version of a requirement parameter (outlined in red).

4.8 Providing on-the-shelf services for the Thermal Architect



Figure 22: Services to the Thermal Architect

Through its process integration capability, Isight enables Method Engineers to build automated simulation services intended to the Thermal Architect and CAE analyst. Complementarily, simulation data management capabilities of SIMULIA V6 Scenario Definition module are used to manage the lifecycle of these services and to deploy them within the en-

terprise. Moreover, it will manage the data relative to each usage of these services.

These simulation services are intended to be generic enough so that they are applicable on a class of design problems, such that, once a service is published by the Method Engineer, this service can be used on different designs without requiring rework by the consumer of this service.

After instantiation by the end user, the V6 impact graph functionality will enable to completely trace the data flow of the simulation data produced by these services, so that the end-users will be able to understand which data contributed to the generation of a particular data. The example below shows the dependency of an optimized design candidate on:

- the parameterized system architecture (data created by the Thermal architect)
- the CFD model used to generate the surrogate model that is fed into the 0-1D modelling (data created by the CAE analyst).

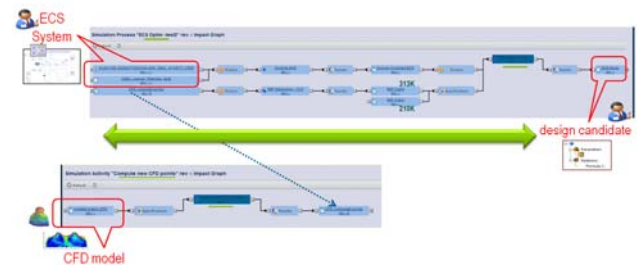


Figure 23: Traceability

4.9 Parameter management

PLM parameters can be defined within the requirements by the Aircraft Architect and reused e.g. within the Logical system by the Thermal Architect.

These PLM parameters, which have a lifecycle of their own and are likely to be revised, can be used to publish requirements characteristics such as expected Cabin temperature range (e.g. between 20 and 24°C), range of operating conditions (e.g. external temperature between -60 and 40°C) that the aircraft can be exposed to, as well as performance targets (e.g. maximum mass).

These PLM parameters are then available downstream in the R-F-L-P cascade, and can be used locally to valuate Knowledgeware parameters that parameterize applicative V6 data like CATIA System Modelica models.

Reuse of Knowledgeware parameters in the Modelica models creates links between parameters in the Modelica models and other data in order to ensure consistency between teams of different engineering disciplines that normally do not have much direct

interaction. An example that we show is how a parameter from the requirements like the external temperature range is reused to drive the values of the external temperature within the alternative models of the Environment of the ECS.

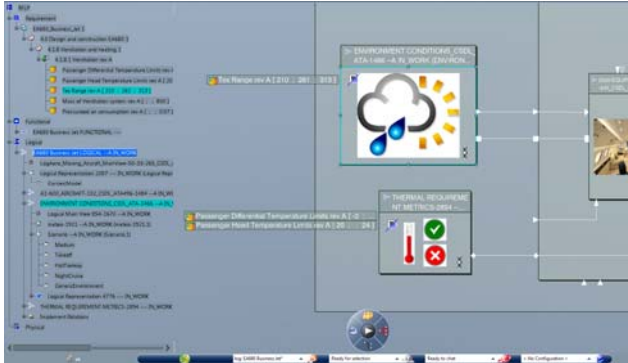


Figure 24: Parameter Flow from Requirements to System

4.10 Decision support interactive environment

In order to identify the design points of interest and to be able to compare these design points, there is a need for a graphical environment that is able to show two complementary views of the engineering problem (cf. figure 25): the analytical view focusing on the performance and constraints (cf. figures 26 and 27), and the behavior centric view that shows, for a specific design point, the associated simulation results (0-1D, CFD, etc...) showing how the virtual product behaves.

This graphical environment is fed with the results generated by design exploration processes mentioned in section 4.8 and is itself packaged as a service to ensure efficiency, consistency and traceability, quite important characteristics for the decisions that will be taken using this environment.

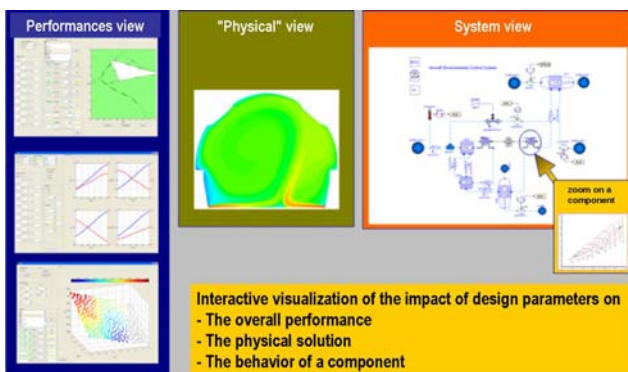


Figure 25: Graphical Environment for Decision Support

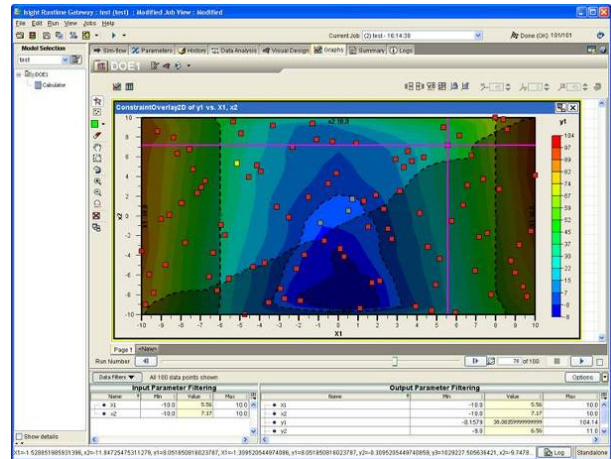


Figure 26: Decision views

In addition, using surrogate model it is possible make interactive request offline. For instance, the feasible domain can be interactively visualized for any combination of design parameters and constraints (cf. figure 27)

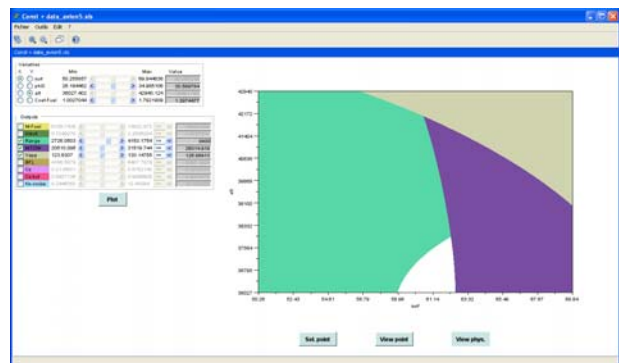


Figure 27: Interactive feasible domain

The ability to performed interactive analysis is a major towards performing an interactive “what if analysis”.

5 Conclusions

In this article, we have tried to sum up what should be a truly efficient tool for aircrafts systems design. A lot of work has been done to obtain a cutting edge tool which includes system management in a PLM framework.

The purpose is to help designers to focus on important problems in a more and more complex context by providing smart tools that allow them to perform their task more efficiently.

For system simulation, Modelica is a key factor. Many enhancements of the language have made it the leading modeling language for physical modeling. Last but not least is the new integration of synchronous semantics in Modelica 3.3 which allows state of art modeling of control systems and digital electrical systems.

Modelica is spreading rapidly in aerospace applications. Even if the language is much more efficient than other languages, there are still some challenges to have efficient simulations involving large hybrid models of complex multi-systems architectures. Most of these challenges will be studied in the next big European project MODRIO (Model Driven Physical Systems Operation).

Acknowledgements

This work was partially supported by the French government through the Systematic CSDL project.

References

- [1] Advances in Product Modelling and Simulation at Dassault Aviation; Lionel de la Sayette; RTO AVT Symposium, April 2002.
- [2] Modelling of System Properties in a Modelica Framework Audrey Jardin, Daniel Bouskela, Thuy, Nguyen, Nancy Ruel (EDF R&D), Eric Thomas, Laurent Chastanet (Dassault-Aviation), Raphaël Schoenig, Sandrine Loembé (Dassault-Systèmes), Modelica Conference 2011.
- [3] Projet CSDL :, Conception en phase amont de systèmes complexes sur la plateforme PLM/SLM de Dassault Systèmes Michel Ravachol, Jean-Baptiste Quincy, NAFEMS Paris, june 2012.
- [4] Modelling of Uncertainties with Modelica Daniel Bouskela, Audrey Jardin (EDF R&D), Zakia Benjelloun-Touimi (IFP Energies nouvelles), Peter Aronsson (MathCore

Engineering), Peter Fritzson (Linköping University) Modelica Conference 2011.

Project Sites

- [5] www.modelica.org
- [6] www.eurosyslib.org
- [7] www.modelisar.com
- [8] www.ITEA2.org / Modrio

Backward simulation - A tool for designing more efficient mechatronic systems

Matthias Liermann

American University of Beirut, Department of Mechanical Engineering

www.aub.edu.lb

matthias.liermann@aub.edu.lb

Abstract

This paper proposes the use of backward simulation with Modelica as a tool to improve system design. The aim is to introduce system simulation into early design stages of mechatronic systems and to use the same software tools and model libraries that are also used in later stages for dynamic analysis and control design. It seems that the necessity of a control design is one of the main obstacles against the use of conventional dynamic system simulation in early design stages. The main benefit of backward simulation is that it does not require an implementation of feedback control.

The backward simulation approach is explained using the example of a servo-hydraulic drive. The paper shows that it can help to significantly reduce the energy consumption of this system. It is possible to simulate typical duty cycles of the drive without the need to redesign the control for each change.

Keywords: backward simulation; forward simulation; model inverse; hydraulics; mechatronics; servo-drive; efficiency optimization; servo drives; design process

1 Introduction

Dynamic system simulation is useful to analyse the dynamic behavior of systems, to design controllers or to determine cumulative system characteristics. Cumulative characteristics, such as for example energy consumption, depend on the definition of a typical load cycle. Simulation can be used to determine the state variables of components of the system for this load cycle and to predict the expected losses and total energy consumption. This is attractive for the designer in the initial stages of system development. A typical example is the design of a servo-drive. The designer has to make many choices in the initial design stages. Choices include the appropriate type of drives, such

as electrical, hydraulic or pneumatic, drive configuration. For each drive type the designer has to size its components. While dynamic simulation could be very helpful in making these choices, it is, however, rarely used. One of the main reasons is the need for control design, which often requires expert knowledge, [8].

Today, engineers use mainly steady state relationships to size components of a mechatronic system. This can be done with spreadsheet calculations. Some manufacturers move to offer specialized software for the dimensioning and analysis of a drive solution such as the SIZER configuration tool [1] for electric drives. Such tools take some dynamic forces into account, but only for predefined, typical scenarios.

This paper presents the method of backward simulation which allows the use of dynamic system simulation to study different system configurations and to size components. Backward simulation in the context of this paper is synonymous with (forward) simulation of the inverse system model. It means that input and output of the simulation are switched and that the direction of computation goes backward from the physical outputs to required control inputs. As will be explained in the paper, the main benefit of backward simulation is the fact that a control does not have to be implemented. Another benefit is, that implemented with Modelica, the backward simulation approach could be used with the same models and simulation tools used for the conventional (forward) simulation approach. This would lead to a better communication between design and control engineers and improve the product development.

The rest of the paper is organized as follows. The role of support of dynamic system simulation in product design and the benefits that are coming with the additional use of backward simulation are described in section 2. Section 3 reviews the use of backward simulation in literature and presents two simple examples to explain the idea and concept. Section 4

presents the mathematical model of a hydraulic drive as an exemplary application of the backward simulation approach. The backward simulation approach is demonstrated by using it to optimize the hydraulic drive efficiency for a certain load cycle in section 5. Section 6 provides the conclusion of this study.

2 Dynamic simulation support in product development

Figure 1 illustrates the conventional support of dynamic simulation in the design of a servo drive in the left flowchart. Usually the design starts by specifying the desired motion and the expected load. This provides the necessary information about required torque and speed which can be used to configure the system and size its components. The designer depends on analytic and empirical design formulas which he can solve for the unknown parameters. This approach makes sure that the hard requirements can be met. But it may be difficult to include some other important aspects of the design, such as average power consumption or required cooling power. These are cumulative characteristics which depend on the average use. For those aspects to be included, respective empirical or analytical design formulas are harder to define. Some sizing tools such as mentioned in [1] can actually compute power consumption for standard drive cycles on the basis of steady state simulation.

Dynamic system simulation is usually used only in later stages of product development. It is used to accelerate commissioning by setting up control hardware with hardware-in-the-loop simulation. It is also used in commissioning or to trouble-shoot unexpected system behavior. The design and implementation of a feedback control is a characteristic part of dynamic simulation, certainly of servo-drives which operate in closed loop. The control design verifies whether the requirement specifications can be met. Once the control is working, also the cumulative characteristics such as power consumption and required cooling can be assessed. If the investigations identify the need to make changes at this stage in product development, it is clear that the costs of making those changes will be high compared to changes made during conceptual design phase.

An alternative design process is shown in the right side of Figure 1. The initial design is found from the same knowledge and experience as in the conventional design process. The main difference is that backward dynamic simulation is used right from the beginning.

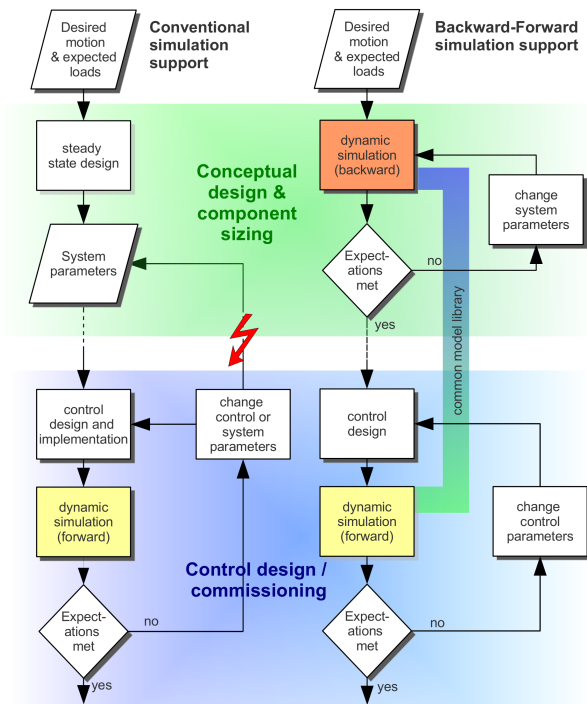


Figure 1: Backward/Forward vs. conventional simulation support of the design process

The (dynamic) simulation model is built from a component library, with the the same models used later for dynamic analysis and control design. However, no control is implemented and the simulation is run in backward mode with the required motion and external forces as boundary conditions. The backward simulation shows if any component runs into physical limitations. Also the energy efficiency over a representative duty cycle can be assessed. Different configurations can be tested to minimize the energy consumption. The backward simulation helps to detect and address dynamic performance issues. As a result it reduces the risk of costly design changes in late stages of the product development. The control design and dynamic analysis of the closed loop can be done at a later stage in the product development. The key advantage of backward simulation for the conceptual design is that a perfect control system is used, where the measured signal is always identical to the desired signal.

Another advantage is that the same dynamic model can be used in later stages of system development for the control design and hardware-in-the-loop simulation. The only difference between backward and forward simulation is in the definition of inputs and outputs and that backward simulation does not need a control to work. Whereas traditionally there is only small overlap of the fields of expertises of design and control engineers, combined backward/forward simu-

lation would enable them to use and update a common tool. This can improve cross-departmental communication and lead to faster and better product development. The use of the backward simulation approach is illustrated in this paper at the example of sizing of a hydraulic servo-axis.

3 Backward simulation

Backward simulation basically is forward simulation of the inverse model. It is to switch cause and effect of a system simulation. The model used for backward simulation is the same model used for forward simulation. The difference is in the definitions of inputs and outputs. The input to the model in forward simulation becomes the output in backward simulation. Forward simulation follows physical principles from cause to effect. Backward simulation can be used to compute the required input for a given output. Backward simulation, or simulation of the model inverse, can be well automated with equation based modeling languages such as Modelica. Dymola, as a simulation tool for Modelica models, is able to calculate the nonlinear model inverse. This capability can be used effectively for system configuration and sizing, but also for nonlinear control. The approach of this paper is closely related to the approach taken in [2], which uses the inverse simulation approach for the optimal selection of drive components in aircraft design. The use of inverse model simulation for nonlinear control schemes is presented in [14, 12].

This paper uses the term backward simulation synonymously for simulating the model inverse. The term backward simulation has been used also by other research groups. The program Advisor, a Matlab/Simulink implementation of model libraries used for optimization of hybrid vehicle drive trains, uses a combined backward-forward simulation approach, [17, 9]. The motivation of using the combined backward-forward approach is to be able to focus on system design. The optimization of drivetrain configurations can be approached without the need for control design. The problem of Advisor is that the way in which the models of this library can be used is predetermined, either backward or forward. Equation based modeling languages such as Modelica have the advantage that the causality of their use is not predefined.

The term backward simulation is also used in the context of backward planning (for example [5]) or for simulation of dynamic systems backward in time (see [6, 13, 16, 15]). In these cases, the simulation aims to

help find the system parameters and initial conditions which lead to a certain result.

3.1 Backward simulation in Modelica

In many components of the Modelica libraries, signal inputs are used, to apply external forces or other constraints or to make changes in component parameters. Signal inputs put restrictions on the connection structure, e.g. an "input" cannot be connected to an "input". It has to be connected to an output. However, it does not define the computational causality as in other approaches like Simulink. A typical example is a hydraulic control valve modeled as a turbulent resistance, see **Figure 2**. The volumetric flow rate Q is propor-

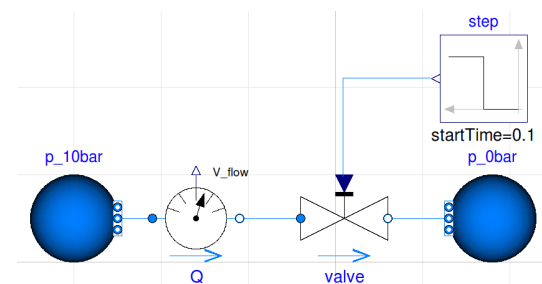


Figure 2: Valve from the Hydraulics library

tional to the partial opening x_V and to the square root of the pressure difference $p_1 - p_2$ across its ports [10]. With the flow gain c_V the flow through a control valve is expressed as

$$Q = c_V x_V \sqrt{p_1 - p_2} \quad (1)$$

Since the square root function is not defined for negative pressure differences and not differentiable for $\Delta p = 0$, often an approximate solution for the square root function is used to implement the flow-pressure relationship in a model [3]. Such a function is implemented as `RegRoot` in the standard Modelica Library in `Modelica.Fluid.Utilities`. It is strictly monotonically increasing, continuously differentiable and therefore invertible.

$$Q = c_V x_V \cdot \text{RegRoot}(p_1 - p_2, \Delta p_{\text{small}}) \quad (2)$$

For the implementation of control valves as for example in `Modelica.Fluid.Valves.ValveIncompressible`, it is assumed that the partial opening x_V is not affected by the pressure difference or flow through a valve. The valve opening is therefore defined as a signal input.

For backward simulation, the partial valve opening x_V needs to be solved for from given flow Q and pressure difference $p_1 - p_2$.

$$x_V = \frac{Q}{c_V \sqrt{p_1 - p_2}} \quad (3)$$

With simulators such as Simulink, where the causality of a model is predefined, assigning pressure and flow as given from boundary conditions leads to an error since the valve opening is defined as a signal input. With Modelica this is possible, as **Figure 3** shows. With the block `Blocks.Math.Inverse-`

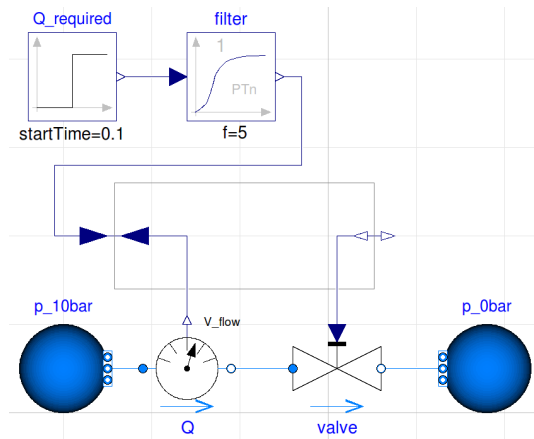


Figure 3: Valve from the Hydraulics library

BlockConstraints it is possible to connect an input function to the volumetric flow sensor and to impose a required flow on the computation while the signal input of the valve can be interpreted as a signal output. What this component does, is, to simply connect the two input signal connectors with each other as well as the two output signal connectors. The effect is that the model inverse is automatically derived by the Modelica translation engine. We see how Modelica allows to simulate the system 'backwards' simply by changing the boundary conditions for inputs and outputs.

3.2 Simple backward simulation example

The input step function in Figure 3 is filtered with a first order filter, without which the simulation would fail. As already stated, backward simulation is forward simulation of the model inverse. Inverting a dynamic model usually requires the derivatives of the input function. This is illustrated at the example of a simple linear system expressed by the transfer function

$$G(s) = \frac{Y(s)}{U(s)} = \frac{1}{s^2 + 2s + 1}. \quad (4)$$

The input-output dynamics written in state differential form is

$$\frac{d}{dt} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -1 & -2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u \quad (5)$$

$$y = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad (6)$$

The inverse of this system can be expressed as transfer function

$$G^{-1}(s) = \frac{U(s)}{Y(s)} = \frac{s^2 + 2s + 1}{1}. \quad (7)$$

However, there is no equivalent expression in state differential form. To express the inverse dynamics, the states would be functions of their derivatives. In Simulink and other assignment based simulation languages, it is important that each model element can be expressed in state differential form. For the inverse dynamics element this is impossible.

The state differential form is required also for the simulation of Modelica models. However, this is reached through automated rearrangement of all sub-model equations. This is a difference to other simulator concepts where each element or sub-model must be represented in state differential form initially.

Figure 4 illustrates how simulation of the inverse model dynamics is possible when the whole system is considered. To implement the simulation of the inverse model, the derivatives of the input to the inverse model must exist. Generating the input through a 2nd order filter assures that two derivatives exist.

$$G_f(s) = \frac{Y(s)}{R(s)} = \frac{25}{s^2 + 10s + 25}. \quad (8)$$

Applying the filtered signal $Y(s) = G_f(s)R(s)$ to the inverse model Eq. 7 yields

$$U(s) = R(s) \cdot \frac{25(s^2 + 2s + 1)}{s^2 + 10s + 25} \quad (9)$$

The combined system can be expressed in state differential form.

$$\frac{d}{dt} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -25 & -10 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 25 \end{bmatrix} r \quad (10)$$

$$u = \begin{bmatrix} -24 & -8 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + 25r \quad (11)$$

Figure 4 shows that the results of backward and forward simulation match. Feeding the forward simulation with the result u of the backward simulations yields the desired system response y that was given as input to the backward simulation.

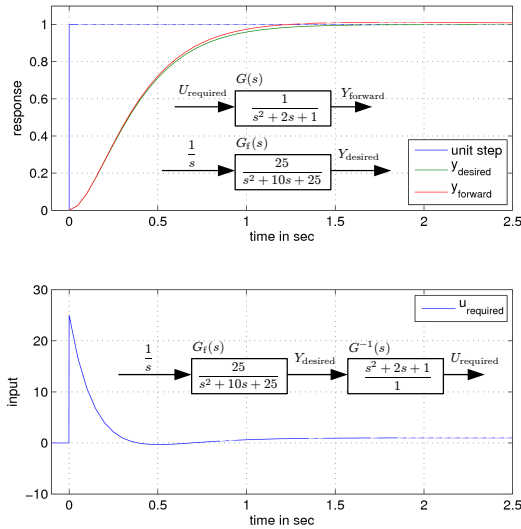


Figure 4: Backward simulation example

3.3 Limitations of backward simulation

The backward simulation approach is limited to systems for which the model inverse exists and is stable. Coulomb friction for example depends on the sign of velocity. For zero velocity the coulomb friction is undefined and depends on the history of motion. This function is not invertible without adjustments. Another challenge are physical limitations implemented in the models. If, during backward simulation, one component reaches a physical limitation, the states which cause the behavior of the model in limitation are not clearly defined anymore. There are infinitely possible combinations of states which cause the limited model to be in its limit. The cases for which the model inverse cannot be obtained are further elaborated in [14, 12]. It is subject of future research to show how relevant these issues are for typical configuration and sizing problems and how they can be addressed appropriately. The next section explains the mathematical model of the example application for which the advantage of the backward simulation approach is demonstrated.

4 Model of example application

A typical model for a servo-hydraulic drive is presented as given in many text books such as [7, 10, 11]. The drive consists of a servo-valve which connects the two ports of a cylinder to a constant pressure supply and a tank, see **Figure 5**.

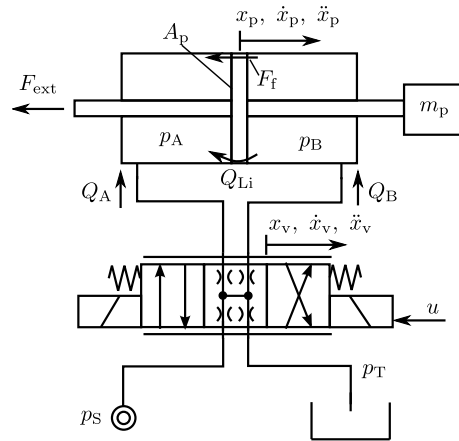


Figure 5: Hydraulic scheme

The model can be described by a system of nonlinear state differential equations of dimension 6.

$$\ddot{x}_p = \frac{1}{m_t(x_p)} [(p_A - \alpha p_B)A_p - F_f(\dot{x}_p) - F_{ext}] \quad (12)$$

$$\dot{p}_A = \frac{1}{C_{h,A}} [Q_A(p_A, x_v) - A_p \dot{x}_p + Q_{Li}(p_A, p_B)] \quad (13)$$

$$\dot{p}_B = \frac{1}{C_{h,B}} [Q_B(p_B, x_v) + \alpha A_p \dot{x}_p - Q_{Li}(p_A, p_B)] \quad (14)$$

$$\ddot{x}_v = -\omega_v^2 x_v - 2D_v \omega_v \dot{x}_v + \omega_v^2 u \quad (15)$$

Where the states and parameters are listed in **Table 1**.

The flow equations are nonlinearly dependent on the valve partial opening x_v and the pressure difference. It has to be defined for different cases depending on which ports are connected with each other.

$$Q_A = c_v \operatorname{sg}(x_v - x_0) \operatorname{sign}(p_S - p_A) \sqrt{|p_S - p_A|} \dots$$

$$\dots - c_v \operatorname{sg}(-x_v - x_0) \operatorname{sign}(p_A - p_T) \sqrt{|p_A - p_T|} \quad (16)$$

$$Q_B = c_v \operatorname{sg}(-x_v - x_0) \operatorname{sign}(p_S - p_B) \sqrt{|p_S - p_B|} \dots$$

$$\dots - c_v \operatorname{sg}(x_v - x_0) \operatorname{sign}(p_B - p_T) \sqrt{|p_B - p_T|} \quad (17)$$

As already explained in section 3.1, the term $\operatorname{sign}(\Delta p) \sqrt{|\Delta p|}$ does not work reliable in a Modelica simulator since there is an infinite derivative whenever Δp becomes zero. For practical implementation the approximate function `RegRoot` can be used. The different switching conditions are realized using the function `sg`, which is defined as:

$$\operatorname{sg}(x) = \begin{cases} 0, & \text{for } x < 0 \\ x, & \text{for } x \geq 0 \end{cases} \quad (18)$$

Table 1: States and parameters of model

Symbol	Comment	Unit
C_h	capacity of chamber	$\frac{m^3}{Pa}$
$E'_{A,B}$	effective bulk modulus	Pa
F_{ext}	external force	N
F_f	friction force	N
$p_{A,B}$	pressure in A,A	Pa
$Q_{A,B}$	flow into chamber A,B	$\frac{m^3}{s}$
Q_{Li}	leakage from chamber B into A	$\frac{m^3}{s}$
u	$\frac{\text{valve signal}}{\text{maximum valve signal}}$	-
$V_{A,B}$	Volume chamber A,B	m^3
x_p	piston position	m
x_v	valve spool partial opening	-
$A_p = 7.6 \cdot 10^{-4}$	piston face side surface area	m^2
$C_{Li} = 1.6 \cdot 10^{-13}$	leakage coefficient	$\frac{m^3}{Pa \cdot s}$
$c_v = 8.9 \cdot 10^{-8}$	valve flow gain	-
$c_s = 0.01$	Stribeck velocity	$\frac{m}{s}$
$D_v = 0.9$	damping ratio of valve	-
$E_{max} = 1.7 \cdot 10^9$	bulk modulus at infinite pressure	Pa
$F_{c0} = 100$	Coulomb friction force	N
$F_{s0} = 100$	Static friction force	N
$m_t = 50$	total mass of piston	kg
$p_s = 200 \cdot 10^5$	supply pressure	Pa
$p_T = 2 \cdot 10^5$	reservoir pressure	Pa
$s = 0.8$	stroke	m
$x_o = -1\%$	fractional valve overlap	-
$\alpha = 1$	piston surface ratio	-
$\gamma = 800$	Approximation factor	-
$\omega_v = 628$	natural undamped frequency of valve	$\frac{rad}{s}$
$\sigma = 1000$	viscous friction coefficient	$\frac{N \cdot s}{m}$

According to the manufacturing of the valve, the spool can have over- or underlap with the sleeve in the middle position. The overlap parameter x_o takes this into account. If it is negative, it means that the valve is underlapped and therefore all valve ports are connected with each other in the middle position of the valve.

The leakage flow across the piston Q_{Li} is the cylinder pressure difference multiplied by the leakage coefficient C_{Li} .

$$Q_{Li} = C_{Li}(p_B - p_A) \quad (19)$$

The pressure gradient \dot{p} in a cylinder chamber is char-

acterized by the hydraulic capacity

$$C_h = \frac{V}{E'} \quad (20)$$

which is the quotient of Volume over effective bulk modulus of the respective chamber. The volumes change with position of the piston, while the bulk modulus varies with the chamber pressure. An empirical model proposed by [4] is

$$E' = E_{max} \left[1 - e^{-0.4 - 2 \cdot 10^{-7} p} \right] \quad (21)$$

where E_{max} is the bulk modulus at infinite pressure.

The friction of the hydraulic cylinder can be represented by the stribeck curve, which is a piecewise defined function.

$$F_f(\dot{x}_p) = \sigma \dot{x}_p + \text{sign}(\dot{x}_p) \left[F_{c0} + F_{s0} e^{-\frac{|\dot{x}_p|}{c_s}} \right] \quad (22)$$

where σ is the viscous friction coefficient, F_{c0} the coulomb friction, F_{s0} the static friction and c_s the so-called Stribeck velocity. Since the friction model as given by Eq. (22) is not invertible, an approximation of it is used by replacing the sign function.

$$\text{sign}(\dot{x}_p) \approx \frac{2}{\pi} \arctan(\gamma \dot{x}_p) \quad (23)$$

And therefore:

$$|\dot{x}_p| \approx \dot{x}_p \frac{2}{\pi} \arctan(\gamma \dot{x}_p) \quad (24)$$

With the values given in Table 1, the friction function is plotted in **Figure 6**.

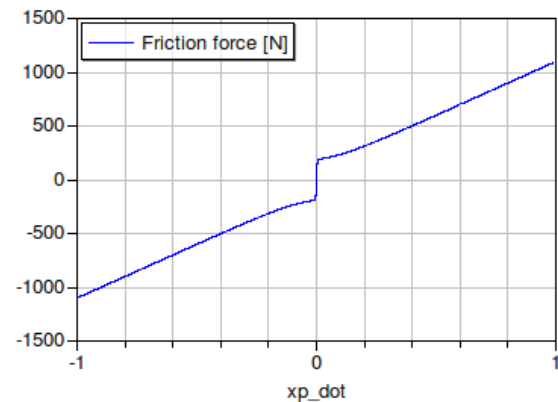


Figure 6: Friction model

The set of differential algebraic equations Eq. (12-15) is given in the standard form of ordinary differential equations, where the state derivatives are expressed as a function of the states and inputs. For

backward simulation the model inverse needs to be expressed by rearranging the equations. This is not possible algebraically in this case. However, Modelica tools, such as Dymola, can generate nonlinear inverse models automatically as explained in section 3. The next section explains how backward simulation can help in sizing a hydraulic system with respect to static and dynamic requirement specifications.

5 Efficiency study using dynamic backward simulation

Backward simulation allows to study the drive's performance for a whole duty cycle without the need to design a controller. In fact, perfect control is assumed because the expected output is forced on the system as a boundary condition. This is an advantage, since sometimes it is the necessity of control design which discourages the early use of system simulation. Often, in early stages of system development, issues such as architecture configuration and component sizing is important. System simulation in forward manner may then be impractical if changes in the system always require re-design of the controller. It is interesting to note that Modelica allows to use the same model for forward and backward simulation. This means that the same model used in backward manner for system configuration and component sizing may be used in forward manner later for the control design.

The idea of backward simulation is to force the prescribed duty cycle as boundary conditions on the physical outputs of the system, see **Figure 7**. Consequently, by simulating the inverse model, the corresponding physical inputs are calculated. To do this, no control has to be implemented. The advantage of this approach is demonstrated at the example of a hydraulic servo drive as modeled in the previous section. The model 'HSS' in **Figure 7** is described by Eqs. 12-15. For the sake of clarity the system was not put together by the commercial Modelon Hydraulics library. The components of this library include some effects which cause problems for the backward simulation approach. For example, an interpolation function is used to calculate the average density within a resistance. This interpolation function causes to fa

Figure 8 shows the required (filtered) duty cycle, the position and velocity trajectories and the external force impact. In this duty cycle the hydraulic drive moves out with a constant velocity of $0.23 \frac{\text{m}}{\text{s}}$ while applying a constant force on a workpiece of 10kN. The return stroke takes place with high velocity of $0.8 \frac{\text{m}}{\text{s}}$.

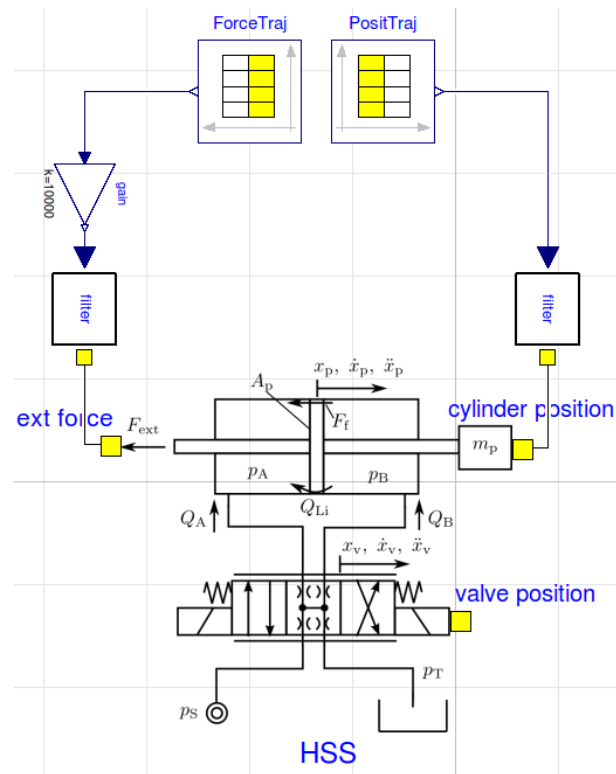


Figure 7: Dymola backward simulation of hydraulic servo system

The objective of optimization is to find the right sizes of cylinder and valve as well as choosing the operating pressure. To do this in conventional forward simulation, a control has to be designed. In case the requirement specifications cannot be met, it is unclear in forward simulation whether the suboptimal control limits the performance or whether the components just don't allow for a better performance.

Simulation of the setup is performed in backward simulation with the parameters as listed in Table 1. The design engineer can examine from the results whether component limitations were violated. It is also possible to examine the total energy consumption. **Figure 9** shows the required valve signal input HSS.xv, the cylinder pressures HSS.pA, HSS.pB and the cumulated amounts of energy E_{loss} , E_{mech} , and $E_{\text{mech+fric}}$. The top plot indicates that the valve size is too small since it opens beyond 100%. The cylinder pressures are within the range between supply and tank pressure. At the beginning of the force impact, the pressure in chamber A has a peak of 185 bar. The load pressure during the working stroke is $\Delta p_{A,B} = (167 - 34) \text{ bar}$ which is 67% of the available pressure difference. According to [11] this is the operating point of optimal efficiency for this type of servo drive. This can be seen well by looking at the bottom plot of

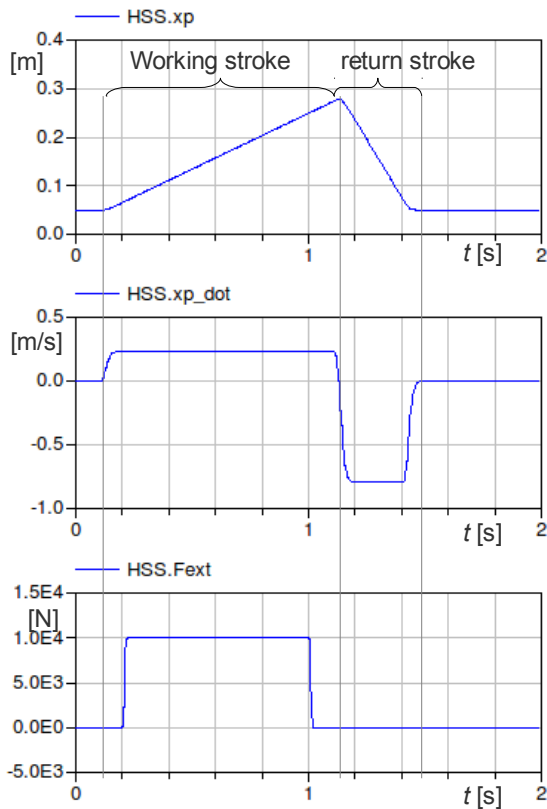


Figure 8: Duty cycle of drive

Figure 9. It compares the total hydraulic input energy E_{loss} , the mechanical output energy E_{mech} and the cumulative curve of mechanical output energy and friction energy $E_{\text{loss+fric}}$. In this example, the friction energy is negligible compared to losses in the valve. During the working stroke, the efficiency is approximately 67% which is optimal according to [11]. The total energy consumption for the working stroke is 3487J. The backward simulation reveals that the energy consumption for the return stroke is equal. This result is interesting although obvious. One might expect that the return stroke should consume less energy because no load is applied. However, the same flow is consumed at the same pressure level. Therefore the power is equal. The backward simulation can now be used to alter the design to achieve a higher efficiency while not violating the valve limitations at the same time.

The losses during the return stroke can be reduced by changing the area ratio of the cylinder. Choosing a faster valve reduces the dynamic peak in the valve and pressure signal at the moment when the load is suddenly applied. Reducing the cylinder area decreases the overall losses and increases the load pressure. The changes according to **Table 2** are found through few

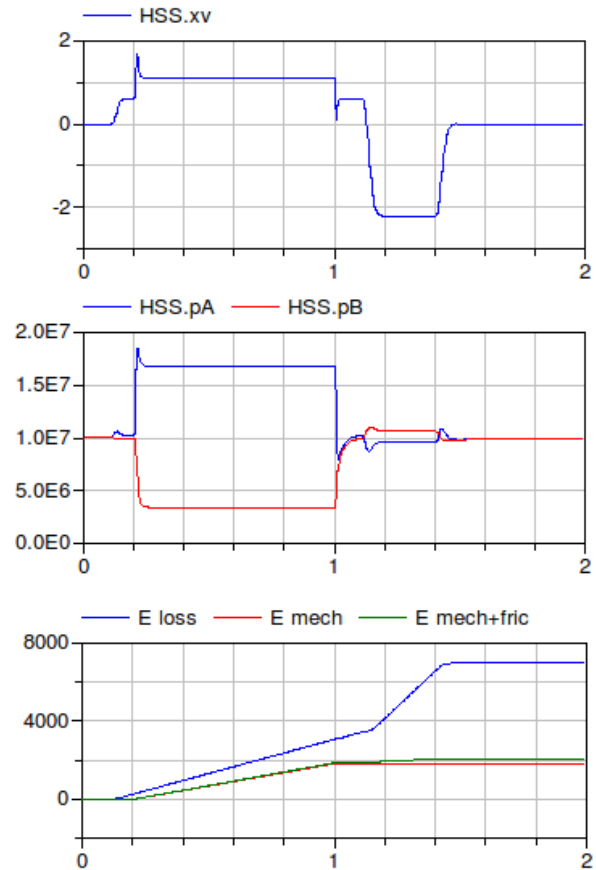


Figure 9: Valve signal, pressures and loss curves for duty cycle

iterative steps and do not represent an optimum solution. But the effect in terms of reduction of energy consumption is significant, as **Figure 10** shows.

The energy consumption of the improved system could be reduced by 38% from 7029J to 4323J for the example duty cycle. The valve was doubled in size and does not run into limitations anymore. This example demonstrates the advantage of the use of backward simulation for the design of servo drives. Through the use of backward simulation the energy efficiency of the systems could be analyzed for a representative duty cycle without the need to design a control. The control

Table 2: Modified parameters

Symbol	Comment	Unit
$A_p = 6.08 \cdot 10^{-4}$	piston face side surface area	m^2
$c_v 17.8 \cdot 10^{-8}$	valve flow gain	-
$\alpha = 0.5$	piston surface ratio	-
$\omega_v = 1256$	natural undamped frequency of valve	$\frac{\text{rad}}{\text{s}}$

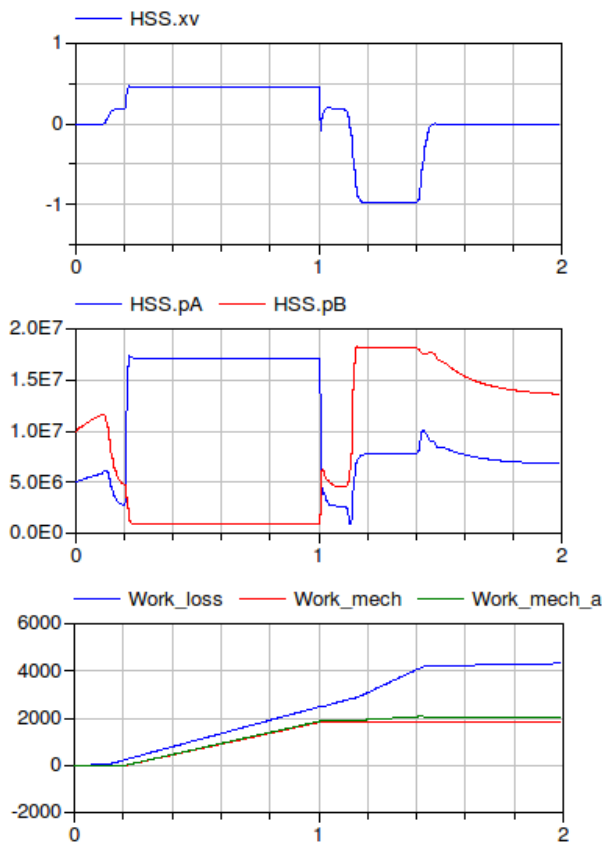


Figure 10: Valve signal, pressures and loss curves for optimized system

design is the next step after the dimensions of the drive have been determined.

6 Conclusion

This paper explains the idea of backward simulation, which is basically forward simulation of the inverse model. It was shown at the example of the mathematical model of a hydraulic servo-drive that building the model inverse by hand is not a trivial task. Depending on the system under study, the model inverse can only be determined numerically. Modelica tools such as Dymola provide this capability and therefore facilitate this new simulation technique. It is explained that the backward simulation approach only works if the system inverse can be build from the model. This may not be possible for systems with backlash or hysteresis. Phenomena like coulomb friction, which are discontinuous, need to be approximated.

The advantage of backward simulation is demonstrated in this paper at the example of a hydraulic servo drive for which a typical duty cycle was given. With little effort, new system parameters are found for

which the energy consumption is reduced by nearly 40%.

This study did not make use of already available libraries.

7 Acknowledgements

The author expresses thanks to the University Research Board of the American University of Beirut for funding this research.

References

- [1] Michael Ambros. Engineeringtool sizer. In *Antriebstag 2011, Siemens Kundenveranstaltung*, 2011.
- [2] Johann Bals, Gerhard Hofer, Andreas Pfeiffer, and Christian Schallert. Object-oriented inverse modelling of multi-domain aircraft equipment systems with modelica. In Peter Fritzson, editor, *Proceedings of the 3 International Modelica Conference, Linkping, November 3-4, 2003*, pages 377–384, 2003.
- [3] 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 5 International Modelica Conference, September 4-5th, 2006*, volume 2, pages 631–640, 2006.
- [4] W. Hoffmann. *Dynamisches Verhalten hydraulischer Systeme, automatischer Modellaufbau und digitale Simulation (Diss.)*. PhD thesis, RWTH Aachen University, 1981.
- [5] Chueng-Chiu Huang and Hsi-Kuang Wang. Backward simulation with multiple objectives control. In *Proceedings of the International MultiConference of Engineers and Computer Scientists 2009 (IMECS)*, volume 2, 2009.
- [6] Amor V.M. Ines and Peter Droogers. Inverse modelling in estimating soil hydraulic functions: a genetic algorithm approach. *Hydrology and Earth System Sciences*, 6(1):49–65, 2002.
- [7] Mohieddine Jelali. *Hydraulic Servo Systems: Modelling, identification and control*. Springer, 2003.

- [8] Matthias Liermann and Hubertus Murrenhoff. Knowledge based tools for the design of servo-hydraulic closed loop control. In *International Symposium on Power Transmission and Motion Control (PTMC)*, pages 17–28, Bath, England, 2005.
- [9] T. Markel, A. Brooker, T. Hendricks, V. Johnson, K. Kelly, B. Kramer, M. O’Keefe, S. Sprik, and K. Wipke. Advisor: a systems analysis tool for advanced vehicle modeling. *Journal of Power Sources*, 110(2):255 – 266, 2002.
- [10] Herbert E. Merrit. *Hydraulic control systems*. John Wiley & Sons, Inc., 1967.
- [11] Hubertus Murrenhoff. *Servohydraulik - geregelte hydraulische Antriebe [Servo-hydraulics - closed loop controlled hydraulic drives]*. Shaker, Aachen, 2008.
- [12] Martin Otter. *Modeling, simulation and control with Modelica 3.0 and Dymola 7.0 (Preliminary Draft, Jan 21, 2009)*. DLR, 2009.
- [13] A. Stohl, C. Forster, S. Eckhardt, N. Spichtinger, H. Huntrieser, J. Heland, H. Schlager, S. Wilhelm, F. Arnold, and O. Cooper. A backward modeling study of intercontinental pollution transport using aircraft measurements. *Journal of Geophysical Research*, 108(D12):Ach 8 1–18, 2003.
- [14] M. Thümmel, G. Looye, M. Kurze, M. Otter, and J. Bals. Nonlinear inverse models for control. In Gerhard Schmitz, editor, *Proceedings of the 4th International Modelica Conference, Hamburg, March 7-8, 2005*, pages 267–279, 2005.
- [15] Christopher D. Twigg and Doug L. James. Backward steps in rigid body simulation. *ACM Transactions on Graphics*, 27:25:1–10, 2008.
- [16] Eric A. Wan, Er A. Bogdanov, Richard Kiebertz, Antonio Baptista, Magnus Carlsson, Yinglong Zhang, and Mike Zulauf. Model predictive neural control for aggressive helicopter maneuvers. In *Software Enabled Control: Information Technologies for Dynamical Systems, chapter 10*, pages 175–200. IEEE Press, John Wiley & Sons, 2003.
- [17] K.B. Wipke, M.R. Cuddy, and S.D. Burch. Advisor 2.1: a user-friendly advanced powertrain simulation using a combined backward/forward approach. *Vehicular Technology, IEEE Transactions on*, 48(6):1751 –1761, November 1999.

Modelling of new vehicle suspension concept with integrated electric drive

Jakub Tobolář^a Jakob Müller^b Alfred Pruckner^b

^aGerman Aerospace Center (DLR), Institute of Robotics and Mechatronics, Germany;

^bBMW Group Forschung und Technik, Munich, Germany

Abstract

In the last decade an electrification of the powertrain became the significant trend in the passenger cars' development. Beside hybrid electric powertrains there is also a variety of solutions for pure electric cars. The presented paper introduces a Modelica model of an electric vehicle solution with rear driven wheels. The suspension model containing an individual electric drive placed close to the wheel will be discussed with focus on different modelling aspects. Moreover, some typical characteristics of suspension will be presented.

Keywords: Modelica; vehicle suspension; planetary gearbox; electric car; connecting multibody with one-dimensional

1 Introduction

In the last years, the electrification of the powertrain of passenger cars became one of the huge challenges for the vehicle developers. This is the consequence of the legislative demand to reduce the emissions and of customer wish to reduce fuel consumption as well.

Several solutions for the hybrid electric vehicles exist such as parallel or serial arrangement of internal combustion engine and electric drive. Whereas such solutions are preferred for mass-production vehicles the pure electric vehicles are still designed in significantly lower series.

Electric cars commonly utilise either vehicle body mounted motors or in-wheel drives. For the first one the gearbox and drive shafts transmit the drive torque to the wheels. An alternative concept for electric car suspension with drive close to the wheel was developed in the joint research project of BMW Group Forschung und Technik, DLR and Schaeffler Group. This concept should utilise advantages of abovementioned common solutions and additionally minimise

the required space needed for all components including battery.

To investigate the behaviour of the suspension in the early design stage the multibody model was created. Later, a model of electric motor was additionally utilized in an overall vehicle model. Based on the vehicle model a drive control strategy was developed and optimized for various drive manoeuvres.

The presented paper focuses on different aspects of the modelling in the early design stage.

2 Suspension concept and design

Typical for BMW vehicles, the developed driven suspension was designed for the rear axle, see the resulting overall design as shown in Figure 1. The basic idea was to couple the electric drive fixed on vehicle body with suspended wheel by means of a gearbox integrated in the wheel, see [1].

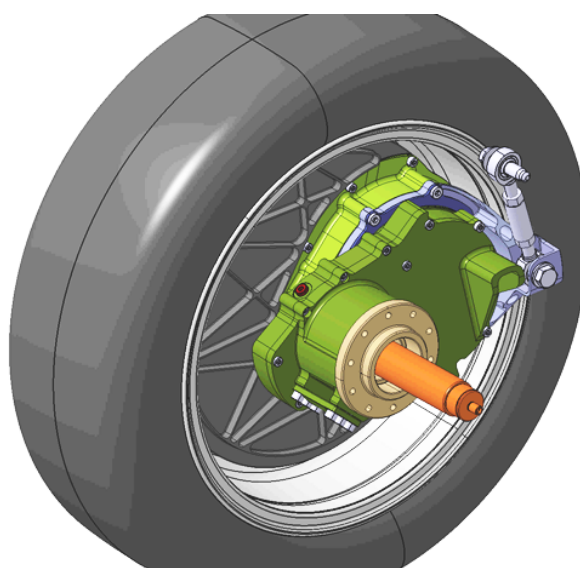


Figure 1: Overall view of the presented rear suspension (electric drive not displayed)

For the wheel guidance there is used a mechanism which can be simply imagined as double pendulum, cf. Figure 2. A swing arm rotating about the axis n_1 oriented in lateral direction is fixed on the vehicle body. The wheel carrier is joined rotationally to this swing arm, whereby the axis n_2 of rotation points to the lateral direction as well. To constrain one redundant degree of freedom, the wheel carrier is additionally linked to the vehicle body. The link is placed before the rotational axis n_2 of the carrier. To tune the kinematic characteristics of the suspension the orientation of the two rotational axes n_1 and n_2 and the position of the link mounting points can be changed.

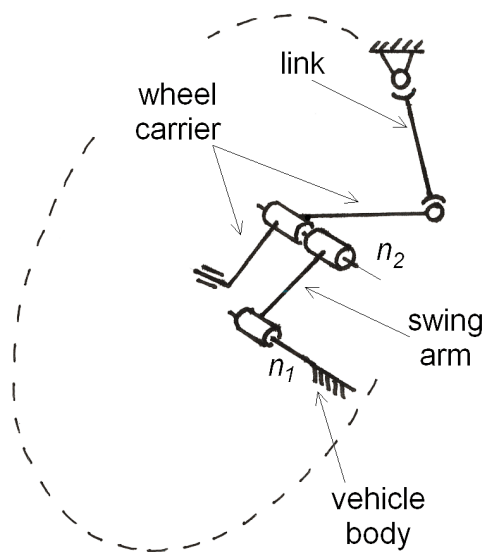


Figure 2: Structure of the wheel guidance mechanism

As mentioned above, the in-wheel gear was suggested for the power transmission to minimise the required space in a vehicle. Due to this solution the electric drive can be placed close to the wheel at the rotational axis n_1 of the swing arm. The driving torque is transmitted from the drive pinion to the lay shaft on axis n_2 and then to the wheel rim. To reach the desired ratio from drive pinion to wheel, there are two “planet” gearwheels on the lay shaft, one in contact with the pinion and the other in contact with the ring wheel.

To minimise required space a rotational damper and spiral spring were employed in suspension. Using such rotational elements was the best way to exploit the large rotational movements of suspension during deflection and rebound. The spring was placed on axis n_1 and designed to react the torques acting on the swing arm. It is supported directly on the vehicle body.

The damper is connected to the swing arm and wheel carrier instead. Placed on the same axis n_2 like the lay shaft, the rotational velocity of the damper is approximately double of that of the spring.

Especially for acoustic reasons, there are used elastic bushings and a subframe, too, as usual in the vehicle design. Each of the suspensions is coupled by means of bushings on the subframe thus constituting one axle unit. Finally, the complete subframe is mounted elastically on the vehicle body.

3 Modelling

The aim of the modelling was to create the multibody model of the suspension and of the complete vehicle to perform common analyses during the early design stage. Additionally, the multibody model was used to tune the vehicle dynamics control.

In order to promote easy interoperability with the various automotive libraries not only from DLR, the created Modelica library was consequently based on the *VehicleInterfaces* standards, see [2]. The *VehicleInterfaces* focuses on standardising the assemblies interface definitions without enforcing a standard vehicle model architecture, so that the same assembly models can be reused in different model architectures. For example, the chassis assembly uses the same interface definition regardless of it being a basic one-dimensional (1D) longitudinal model or a complex multibody vehicle dynamics model.

All the assembly models were created based on the idea of template and parametrised models as also utilised in the *PowerTrain* library from DLR, see [3], [4] and [5]. Therefore, every Modelica subpackage with assemblies such as suspensions or steerings contains template models – i. e. assembly models of different level of detail and for diverse purposes. Various meaningfully parametrised models of realistic assemblies are then inherited from such template models and used in an overall vehicle architecture thus representing a particular vehicle model.

In the following the modelling of the discussed suspension will be described in more detail.

3.1 Wheel guidance

The wheel guidance mechanism was realised as a multibody model with two rigid bodies and two rotational joints, each one for the swing arm and for the wheel carrier. The movement of the wheel carrier was constrained with the link modelled as the *Universal-*

Spherical joint from Multibody package of Modelica standard library. The final design of the suspension utilizes slightly skew rotational axes to achieve suitable kinematics common for rear axles of passenger cars, see camber and toe angle characteristics in Figure 3.

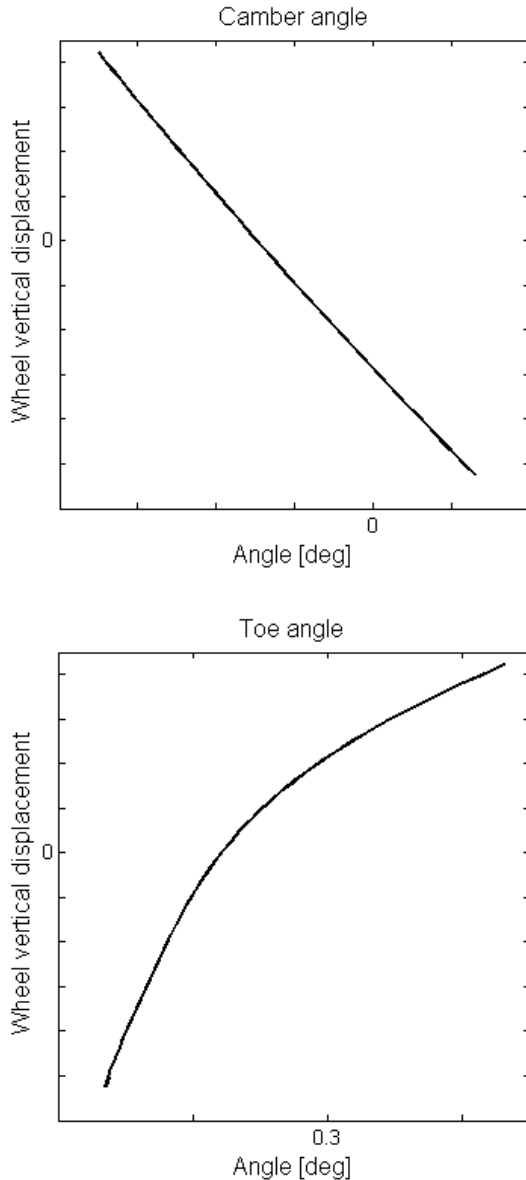


Figure 3: Camber and toe angle of suspension

3.2 Gearbox

The two-stage gearbox was modelled by means of two one-dimensional rotational *PlanetPlanet* models (see [5]) from *PowerTrain* library, cf. blocks *sunPlanet* and *planetRing* in Figure 4. The *sunPlanet* represents the first stage from drive pinion to the lay shaft, the *planetRing* is used for the second stage from lay

shaft to the ring gearwheel which is fixed on the wheel rim. This configuration is similar to that of a planetary gearbox.

In the multibody model of the suspension the rotational joints of the wheel as well as of the wheel carrier connect the respective body to the predecessor one. It means, that the joint angles reflect relative rotation of wheel to wheel carrier and of wheel carrier to swing arm, respectively. On the contrary, the angles of sun and planet gearwheels and of carrier in the *PlanetPlanet* model are absolute angles as adequate for a 1D rotational mechanics. Therefore, a 1D sub-model was added which calculates necessary relative angles to connect the 1D gearbox and multibody suspension in a correct way. This sub-model was called *RotationalAdd* and used two times in the gearbox model, see the blocks called *rotAdd2_i*, $i = 1, 2$ in Figure 4.

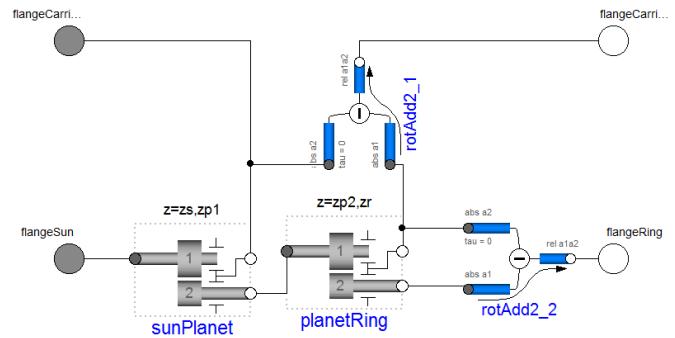


Figure 4: One-dimensional Modelica model of the gearbox

This sub-model for rotational addition has three flanges: 1, 2 and 12. The corresponding flange angles ϕ_1 , ϕ_2 and ϕ_{12} result from simple kinematics:

$$\phi_2 = \phi_1 + \phi_{12}.$$

The appropriate flange torques yield

$$\tau_1 = 0, \quad (1)$$

$$\tau_2 \omega_2 + \tau_{12} \omega_{12} = 0. \quad (2)$$

The equations (1) and (2) both summarise power balance on flanges 1, 2 and 12. Herewith, the power flow between 2 and 12 must be in balance. At the flange 1, in contrast, no power flow may be realised.

Let us focus now on the connection of the *sunPlanet* component as first stage of the gearbox to the swing arm joint. The swing arm body is identical with the carrier of *sunPlanet*. For swing arm joint the revolute joint from Modelica standard library called *Modelica.Mechanics.MultiBody.Joints.Revolute* could

be used, cf. [6]. This revolute joint has two multibody (a and b) and two 1D rotational ($axis$ and $support$) connectors. Simplified described, the following torque balance is adopted between the connectors:

$$\begin{aligned}\tau_{MBS,a} &= -\mathbf{T}_{ab} \tau_{MBS,b}, \\ \tau_{1D,axis} &= -\tau_{MBS,b} \mathbf{n}_{axis},\end{aligned}$$

with transformation matrix \mathbf{T}_{ab} from multibody frame b to frame a and vectors of cut torques $\tau_{MBS,a}$ and $\tau_{MBS,b}$ in frames a and b , respectively, and normalised vector \mathbf{n}_{axis} of rotation axis. It should be noted that the torque from 1D flange $support$ does not apply within this joint. Consequently, when connecting 1D flange $axis$ to $flangeCarrier$ from 1D gearbox (see connectors in Figure 4) the scalar carrier torque $\tau_{flangeCarrier} = \tau_{1D,axis}$ would be applied on multibody frame b and supported on frame a . However, this is unrealistic since for idealised frictionless joints the supporting torque can only be realised by the torque source, i. e. by the electric drive.

Therefore, the rotational joint equations are modified as follows:

$$\begin{aligned}\tau_{MBS,a} + \tau_{1D,support} \mathbf{n}_{axis} &= -\mathbf{T}_{ab} (\tau_{MBS,b} + \tau_{1D,axis} \mathbf{n}_{axis}), \\ \tau_{1D,axis} &= -\tau_{MBS,b} \mathbf{n}_{axis}.\end{aligned}$$

With such a definition, the torque from 1D flange $axis$ only applies on multibody frame b and the torque from 1D $support$ on frame a . Consequently, the carrier torque from 1D gearbox only applies on the swing arm and not on the predecessor body. The connection of the second stage of the gearbox to the wheel carrier joint is arranged in the same way.

The final connection of the total gearbox model with the multibody components is depicted in Figure 5.

3.3 Spiral spring and damper

For suspension the rotational spiral spring was connected between vehicle body and swing arm. It was designed to optimally support reaction forces and torques acting in the swing arm mounting. Especially, the torque about the swing arm rotational axis and the vertical force were considered, both resulting from tyre/road contact. The rotational stiffness of the realised spring is nearly constant over the whole wheel deflection range.

Both cut torques and cut forces on the spring mounts are dependent on their relative deflection, i. e. their relative orientation and displacement. These dependencies were modelled by means of multi-dimensional tables within one multibody force element. The tables

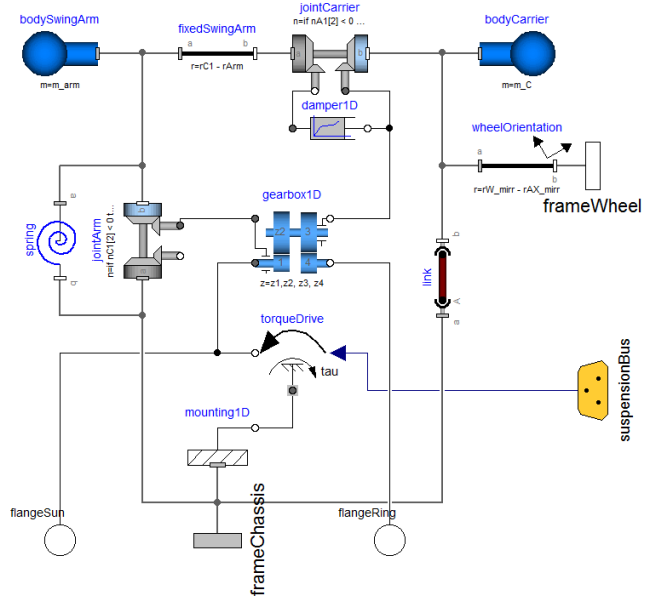


Figure 5: Modelica model of suspension (some marginal elements are not shown)

were generated previously on the base of a finite element spring model.

The rotational damper acting between swing arm and wheel carrier was modelled as one-dimensional non-linear damper.

3.4 Axle subframe and bushings

On the rear axle, each of the suspensions is mounted on a module carrier. Each of these module carriers is then elastically mounted on a subframe which again is elastically mounted on the vehicle body.

The elastic bushings in the mounts are modelled as force elements. Generally, it is formulated in such a way that the forces and torques depend linearly on the relative position and orientation angles of its connectors, respectively, and on their derivatives. The bushings operate at small angles, i. e. at angles with a magnitude less than 5° . This fact was considered to simplify the calculations.

4 Suspension kinematics and brake support angle

In our project, the functionality of the suspension was proven in various tests, both virtual and real. Within this section the suspension kinematic characteristics will be discussed in more detail.

As already shown in Figure 3, the progress of camber and toe angles was tuned when modifying the orientation of rotational axes n_1 and n_2 of the swing arm and the wheel carrier, respectively.

Let us now exploit next typical characteristics - the support angle ε_B at braking. Together with the brake support angle of front suspension and the height of centre of gravity of the vehicle it is crucially responsible for the amount of vehicle pitch movement during braking. According to [7], this angle can be calculated by means of the translational velocity \mathbf{v}_W^* at the “virtual” tyre/road contact point as

$$\tan \varepsilon_B = \pm \frac{v_{Wx}^*}{v_{Wz}^*}.$$

Such virtual contact point is considered on the wheel with blocking brake during wheel deflection and rebound. In a case of conventional vehicle with brakes mounted on the wheel carrier this means that the wheel can be virtually fixed on the carrier during computer aided investigation of ε_B . For practical reasons the point can simply be considered to be on the wheel carrier, too. On the contrary, when the brake is mounted otherwise, e. g. on the vehicle body, such virtual blocking must be regarded in a correct way.

This is also the case for the described suspension. Since the suspension is considered to have no conventional friction brake, the brake torque will only be realised via electric drive. Therefore, in the simulation the drive pinion was fixed for the brake support angle analysis. In Figure 6 the trajectory of the virtual contact point is depicted for our case compared to the point trajectory at conventionally braked wheel.

The analysis proved that the brake support angle ε_B depends not only on the suspension geometry but additionally on the gear ratio i_{Total} as depicted by means of three curves in Figure 7 for increasing ratio.

5 Conclusions

The paper gives an overview of the new vehicle suspension concept with integrated gearbox and electric drive and focuses especially on modelling aspects. In the model, the multibody suspension parts are combined with one-dimensional rotational elements for two-stage gearbox thus enabling efficient simulation. For proper interaction between such one-dimensional and multibody parts new Modelica models were introduced.

Besides the wheel guidance functionality, the particular models of gearbox, spiral spring and bushings are

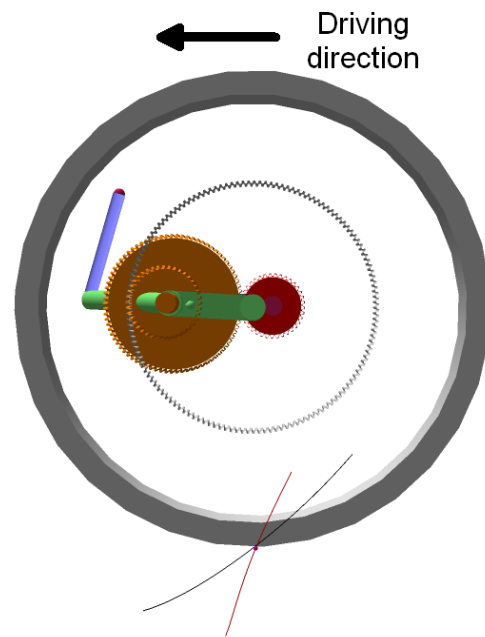


Figure 6: Trajectory of virtual tyre/road contact point of developed suspension (suspension representation simplified): Braking via electric drive (red line at the bottom) vs. conventional brake mounted on wheel carrier (black line)

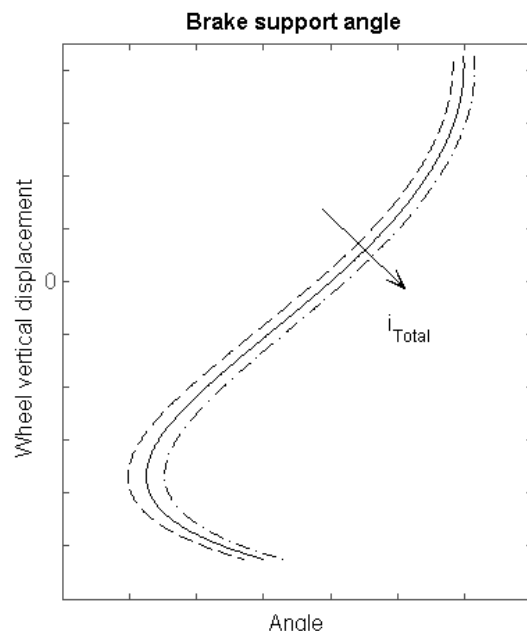


Figure 7: Brake support angle of suspension for varying gear ratio i_{Total}

discussed. Finally, some particular kinematic characteristics of the suspension are discussed in more detail.

The incorporation of the suspension into the complete vehicle model and the comparison of the simulation results with the real driving manoeuvres as well as the utilised drive control strategy will be addressed in the future.

6 Acknowledgements

The development of the presented suspension with integrated electric drive was partly supported by “*Bayerische Forschungstiftung*” under contract AZ-840-08 for the project *Fahrwerk/Antrieb Integration ins Rad (FAIR)*.

References

- [1] Pruckner A.: Potenziale eines radnahen Elektroantriebs zur Gestaltung neuer Antriebs- und Fahrzeugarchitekturen. VDI-Fachkonferenz Berechnung und Erprobung bei alternativen Antrieben. Baden-Baden, 2011
- [2] Dempsey M., Gäfvert M., Harman P., Kral Ch., Otter M. and Treffinger P.: Coordinated automotive libraries for vehicle system modelling. In: Proceedings of the 5th International Modelica Conference. Vienna, 2006.
- [3] Tobolář J., Otter M., Bünte T.: Modelling of Vehicle Powertrains with the Modelica PowerTrain Library. In: Proceedings of the Dynamisches Gesamtsystemverhalten von Fahrzeugantrieben. Haus der Technik Essen, Augsburg, 2007.
- [4] Schweiger Ch., Dempsey M. and Otter M.: The PowerTrain Library: New Concepts and New Fields of Application. In: Proceedings of the 4th International Modelica Conference. Hamburg–Harburg, 2005.
- [5] Pelchen Ch., Schweiger Ch., Otter M.: Modeling and Simulating the Efficiency of Gearboxes and of Planetary Gearboxes. In: Proceedings of the 2nd International Modelica Conference. Oberpfaffenhofen, 2002.
- [6] Otter M., Elmqvist H., Mattsson S. E.: The New Modelica MultiBody Library. In: Proceedings of the 3rd International Modelica Conference. Linköping, 2003.
- [7] Matschinsky W.: Radführungen der Straßenfahrzeuge. 3rd Edition, ISBN 978-3-540-71196-4, 2007.

Dynamic modeling and simulation of a multi-effect distillation plant

Lidia Roca¹ Luis J. Yebra¹ Manuel Berenguel² Alberto de la Calle¹

¹CIEMAT-PSA, Ctra. Senés s/n, 04200 Tabernas, Almería, Spain

Centro de Investigaciones Energéticas, MedioAmbientales y Tecnológicas
Plataforma Solar de Almería

²Dep. Lenguajes y Computación, Universidad de Almería,
Ctra. Sacramento s/n, 04120 Almería, Spain

{lidia.roca,luis.yebra,alberto.calle}@psa.es,beren@ual.es

Abstract

This paper describes a model which simulates the dynamics of a multi-effect distillation system in different operating conditions. It has been designed to improve the operation of the process and develop a control strategy which optimizes the distillate production. The physical models are based on conservation equations of mass and energy. They also include experimental correlations for heat transfer coefficients. Conservation laws are applied in different components, such as the heater, the effects and the preheaters. The results of the mathematical model simulation of the whole process show promising outcomes.

Keywords: solar desalination, multi-effect distillation, modeling

1 Introduction

One of the challenges today is the production of freshwater for those population areas with high water stress. For places close to the sea, the desalination process provides an excellent way to tackle this problem. The use of desalination plants in these regions with plentiful seawater resources is becoming a technological way to produce freshwater. Since large-scale desalination typically requires large amounts of energy, a solution is coupling desalination plants with renewable energies [10]. This process can be performed in various ways, for instance, using solar energy in which the source that provides the heat for the desalination process is collected in a solar field.

Multi-effect distillation plants (MED) raise a great interest in industry due to its efficiency when they are coupled with a solar thermal system. This kind of sys-

tems is gaining more acceptance as a result of their lower energy requirements, higher heat transfer coefficients, compactness, high product water quality and low pre-treatment [2, 7]. In the literature there is a wide variety of steady-state models for MED plants [3, 5, 6, 9]. One of the last works is the one developed in [13], which shows a hybrid system that combines a desalination system with solar and wind energies. In that paper, the model includes the distillation unit, the flat-plate collectors and the wind system. Regarding dynamic models, the literature about multi-effect distillation systems is scarce [4, 8].

The innovation of the present paper is that the dynamic model has been developed with the object-oriented Modelica language using the Dymola tool and the *Modelica.Thermal* library. This framework has allowed us to develop new libraries to make simulations easier and improve the operating procedure.

2 Description of the system

The AQUASOL system (Figure 1) at CIEMAT-Plataforma Solar de Almería (PSA), located in the South of Spain, proposes a solar distillation technology that consists of a compound parabolic collector (CPC) solar field, two 12m³ water storage tanks, a multi-effect distillation unit with a 3m³/h nominal distillate production, and a double effect (LiBr-H₂O) absorption heat pump (DEAHP) [1].

The desalination plant at CIEMAT-PSA is a forward-feed multi-effect distillation unit manufactured and delivered by Weir ENTROPIE (Paris, France) in 1987. It has 14 cells, or effects, in a vertical arrangement. The original first cell that worked with low-pressure saturated steam (70 °C, 0.31 bar [11])

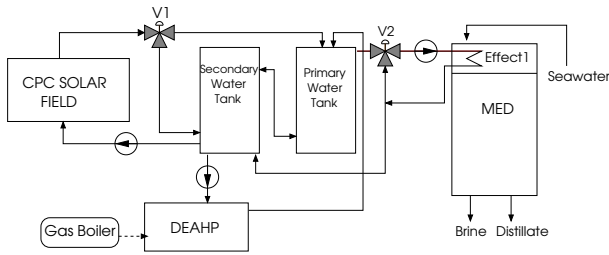


Figure 1: AQUASOL diagram

was replaced in the AQUASOL project by a new one, which works with hot water coming directly from a thermal storage tank. For optimal operation, the inlet feed-water temperature in the first cell must be around 66.5°C. It is possible to reach this temperature with heat from a solar field as well as with steam generated by an auxiliary gas boiler coupled to a double effect absorption heat pump that can work at variable steam loads (from 30% to 100%).

Seawater is preheated on its way toward the first cell of the plant, which is at the top of the desalination tower. Vapour is produced in this first effect (or heater) using the hot water from the storage system. This vapour flows to the preheater-1 and part of the latent heat is transferred to the seawater that flows inside this preheater, increasing the temperature of the seawater. The steam produced in the first effect goes to the effect-2, where it is condensed in a tube bundle sprayed with the more concentrated brine which falls by gravity from the previous effect. The latent heat released by condensation of the vapour allows part of the seawater entering the second effect to evaporate at a lower temperature/pressure. This condensation/evaporation process is repeated in the successive effects. Finally, the vapour produced in the effect-14 is condensed in a final condenser cooled by seawater.

3 The dynamic model

The model of the MED unit is based on the following assumptions:

- no heat losses with the atmosphere,
- no flash vapour is produced,
- the final distillate production is the sum of the distillate produced in each effect,
- the temperature drop in each effect is equal to the temperature difference in the preheaters,

- the preheater-14 is considered as the final condenser.

Each component of the plant (the heater, the effects and the preheaters) has been modeled in the *Modelica* language using *Modelica.Thermal* library. Figure 2 shows the final model of the MED unit, which includes the heater, 13 effects and 14 preheaters.

The inputs of the model are the feedwater mass flow rate to the heater, \dot{m}_M , the inlet temperature to the heater, T_{iM} , the salt concentration of the seawater coming into the heater, C_{B0} , the pressure in each effect, the seawater mass flow rate, \dot{m}_{sw} , and the inlet seawater temperature to the preheaters 14 and 13. The outputs of the model are the outlet temperature from the heater, T_{oM} , and the distillate production, \dot{m}_d . The correlations of the heat transfer coefficients included in the models were obtained from experiments carried out in the real plant [12]. Nomenclature and subscripts are shown in Tables 1, 2.

Table 1: Nomenclature

Name	Description	Units
A	Surface area	m^2
BPE	Boiling Point Elevation	K
C	Concentration	%
C_p	Specific heat capacity	J/kgK
dT	Temperature difference between successive effects	K
h	Specific enthalpy	J/kg
\dot{m}	Mass flow rate	kg/s
M	Mass	kg
P	Pressure	Pa
Q	Heat transfer rate	W
T	Temperature	K
U	Overall heat transfer coefficient	W/m^2K
λ	Latent heat of vaporization	J/kg

Table 2: Subscripts

Name	Description
B	Brine
d	Distillate
e	Effect
h	Heater
i	Inlet
k	Effect identification number
M	MED heating water
o	Outlet
p	Preheater

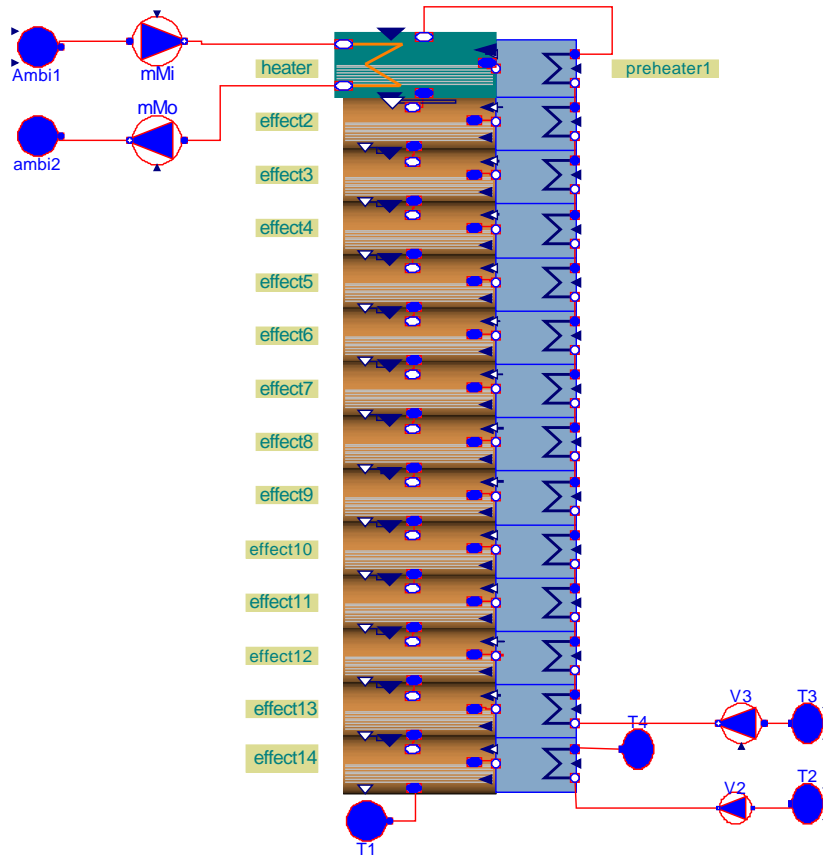


Figure 2: Modelica model of the MED unit

sw	Seawater
v	Saturated vapour

Distillate mass flow produced in the heater can be estimated using the latent heat of vaporization, λ :

$$\dot{m}_{dh} = \frac{Q_h}{\lambda} \quad (3)$$

3.1 The heater

The first effect of the MED plant is the heater. Hot water coming from a storage system enters the heat exchanger and produces the first evaporation of the seawater. Fig.3 shows the model of the heater.

The heat transfer rate for the first effect can be calculated from the MED heating water mass flow rate, \dot{m}_M , and the MED heating water temperature difference in stationary conditions as follows:

$$Q_h = \dot{m}_M \cdot C_p \cdot (T_{iM} - T_{oM}) \quad (1)$$

Using the log-mean temperature and the overall heat transfer coefficient, U_h , the heat transfer rate can be written as:

$$Q_h = U_h \cdot A_h \cdot \frac{(T_{iM} - T_{v1}) - (T_{oM} - T_{v1})}{\ln \frac{T_{iM} - T_{v1}}{T_{oM} - T_{v1}}} \quad (2)$$

where T_{v1} is the saturation temperature of the vapour generated in the heater.

Since the vapour pressure of the aqueous solution is lower than that of pure water at the same temperature, the boiling point of the solution will be higher than that of the water. Therefore, the temperature of the brine can be obtained using the boiling point elevation, BPE:

$$T_{B1} = T_{v1} + BPE \quad (4)$$

The BPE is a brine property and depends on the brine salinity and temperature.

The mass flow rate and concentration of the brine can be obtained applying mass and energy balances.

Mass balance:

$$\frac{d}{dt}(M_{B1}) = \dot{m}_{sw} - \dot{m}_{B1} - \dot{m}_{d1} \quad (5)$$

Salt mass balance:

$$\frac{d}{dt}(M_{B1} \cdot C_{B1}) = \dot{m}_{sw} \cdot C_{B0} - \dot{m}_{B1} \cdot C_{B1} \quad (6)$$

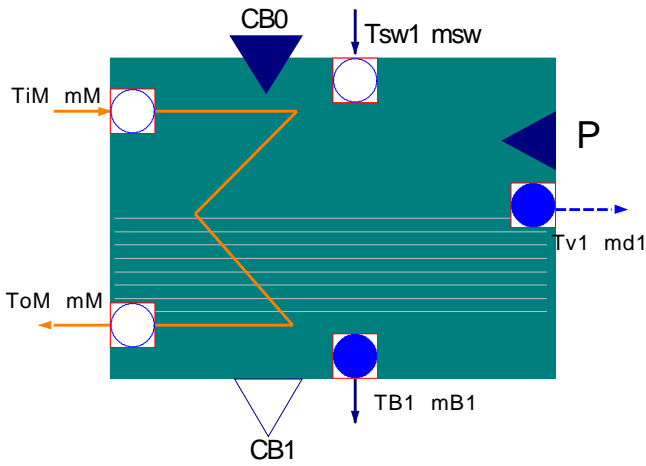


Figure 3: Modelica model of the heater

Energy balance:

$$\frac{d}{dt}(M_{B1} \cdot h_{B1}) = \dot{m}_{sw} \cdot h_{sw} - \dot{m}_{B1} \cdot h_{B1} - \dot{m}_{d1} \cdot h_{v1} \quad (7)$$

3.2 The preheaters

The vapour produced in the heater flows to the preheater-1 located besides, where it condenses as the temperature of the seawater that flows inside the preheater tubes increases. This process is repeated in the successive effects and preheaters. Figure 4 shows the model of the preheater.

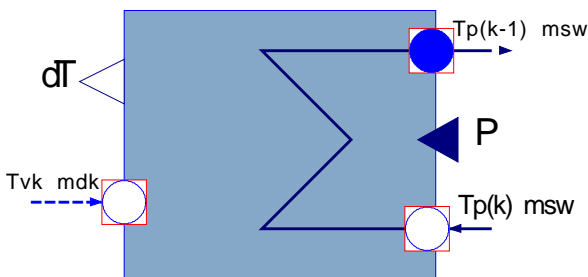


Figure 4: Modelica model of the preheater

The heat transfer rate for each k -preheater was calculated using the measured seawater mass flow rate, \dot{m}_{sw} , and the temperature difference between the outlet and the inlet:

$$Q_{pk} = \dot{m}_{sw} \cdot C_p (T_{p(k-1)} - T_{pk}) \quad (8)$$

Using the overall heat transfer coefficient:

$$Q_{pk} = U_{pk} \cdot A_p \cdot \frac{(T_{p(k-1)} - T_{vk}) - (T_{pk} - T_{vk})}{\ln \frac{T_{p(k-1)} - T_{vk}}{T_{pk} - T_{vk}}} \quad (9)$$

3.3 The effects

The vapour that has not been condensed in the preheater flows to the following effect, where the seawater with a higher brine concentration flows by gravity from the previous effect. Then, the vapour condenses and transfers its latent heat to the seawater producing a new evaporation. Fig.5 shows the model of one effect.

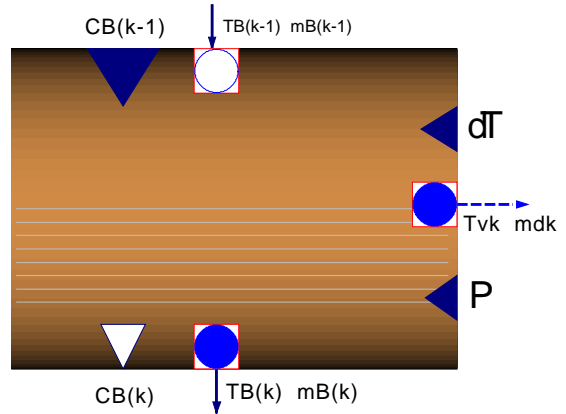


Figure 5: Modelica model of the effect

The heat transfer rate equation for each k -effect evaporator is:

$$Q_{ek} = U_{ek} \cdot A_e \cdot (dT_k + BPE) \quad (10)$$

where dT is the temperature difference between successive effects, which is calculated in the preheater component.

The distillate mass flow rate in the k -effect is:

$$\dot{m}_{dk} = \frac{Q_{ek}}{\lambda} \quad (11)$$

The model of each cell or effect is based on mass and energy balances taking into account the distillate produced and the brine mass flow rate from the previous cell:

$$\frac{d}{dt}(M_{Bk}) = \dot{m}_{B(k-1)} - \dot{m}_{Bk} - \dot{m}_{dk} \quad (12)$$

$$\frac{d}{dt}(M_{Bk} \cdot C_{Bk}) = \dot{m}_{B(k-1)} \cdot C_{B(k-1)} - \dot{m}_{Bk} \cdot C_{Bk} \quad (13)$$

$$\frac{d}{dt}(M_{Bk} \cdot h_{Bk}) = \dot{m}_{B(k-1)} \cdot h_{B(k-1)} - \dot{m}_{Bk} \cdot h_{Bk} - \dot{m}_{dk} \cdot h_{vk} \quad (14)$$

4 Simulation results

The developed model can be used to improve the operation of the plant, studying the effect of the variation in the operating conditions on the MED unit performance and production rate.

The final distillate production will be the sum of the amounts of vapour produced in each effect as follows:

$$\dot{m}_d = \dot{m}_{dh} + \sum_{k=2}^{k=14} \dot{m}_{dk} \quad (15)$$

Figure 6 shows the results obtained simulating the developed model and using the following inputs:

- MED heating water mass flow rate, \dot{m}_M , is 12 kg/s,
- MED inlet heating water temperature, T_{iM} , varies between 338 and 345 K (as shown in Fig. 6)
- seawater mass flow rate inside preheaters 1-13, \dot{m}_{sw} , is 1.94 kg/s,
- preheater-13 inlet seawater temperature, T_{p14} is about 303 K (see Fig. 6).

As it can be observed in Fig. 6, the MED outlet heating water temperature, T_{oM} , is about 3.3 K less than T_{iM} . Nevertheless, if the inlet temperature increases, this difference also increases slightly. As it was expected, higher temperatures cause higher thermal consumption. On the other hand, the higher the inlet temperature, the more distillate is produced.

Therefore, the model may be an efficient tool to estimate the thermal consumption depending on the distillate demand. This means that we can predict if the solar resource is enough to reach the production goals or if we should combine it with the use of the heat pump.

5 Conclusions

In this paper, a multi-effect distillation unit has been modeled. Physical equations for each one of the main components (the heater, the effect and the preheater) have been developed using the object-oriented Modelica language. The whole plant has been defined with multiple instances of the effect and preheater subsystems properly interconnected between them. First simulation results are promising and the model may be used to improve the operation in the real plant. The main purpose of the model is the prediction of the thermal dynamics of the heater as well as the prediction of the distillate production rate.

Acknowledgements

The authors would like to thank the CIEMAT Research Centre and the Spanish Ministry of Economy and Competitiveness for funding Project DPI2010-21589-C05-02.

References

- [1] D. Alarcón-Padilla, L. García-Rodríguez, and J. Blanco-Gálvez. Assesment of an absorption heat pump coupled to a multi-effect distillation unit within aquasol project. *Desalination*, 212:303–310, 2007.
- [2] M.T. Ali, H.E.S. Fath, and P.R. Armstrong. A comprehensive techno-economical review of indirect solar desalination. *Renewable and Sustainable Energy Reviews*, 15(8):4187–4199, 2011.
- [3] N.H. Aly and A.K. El-Figi. Thermal performance of seawater desalination systems. *Desalination*, 158(1-3):127–142, 2003.
- [4] N.H. Aly and MA Marwan. Dynamic response of multi-effect evaporators. *Desalination*, 114(2):189–196, 1997.
- [5] H. El-Dessouky, I. Alatiqi, S. Bingulac, and H. Ettouney. Steady-state analysis of the multiple effect evaporation desalination process. *Chemical engineering & technology*, 21(5):437, 1998.
- [6] A.M. El-Nashar. Predicting part load performance of small med evaporators—a simple simulation program and its experimental verification. *Desalination*, 130(3):217–234, 2000.
- [7] M.A. Eltawil, Z. Zhengming, and L. Yuan. A review of renewable energy technologies integrated with desalination systems. *Renewable and Sustainable Energy Reviews*, 13(9):2245–2262, 2009.
- [8] A. Husain. *Integrated Power and Desalination Plants*. EOLSS Publishers Ltd., 2003.
- [9] MH Khademi, MR Rahimpour, and A. Jahanmiri. Simulation and optimization of a six-effect evaporator in a desalination process. *Chemical Engineering and Processing: Process Intensification*, 48(1):339–347, 2009.

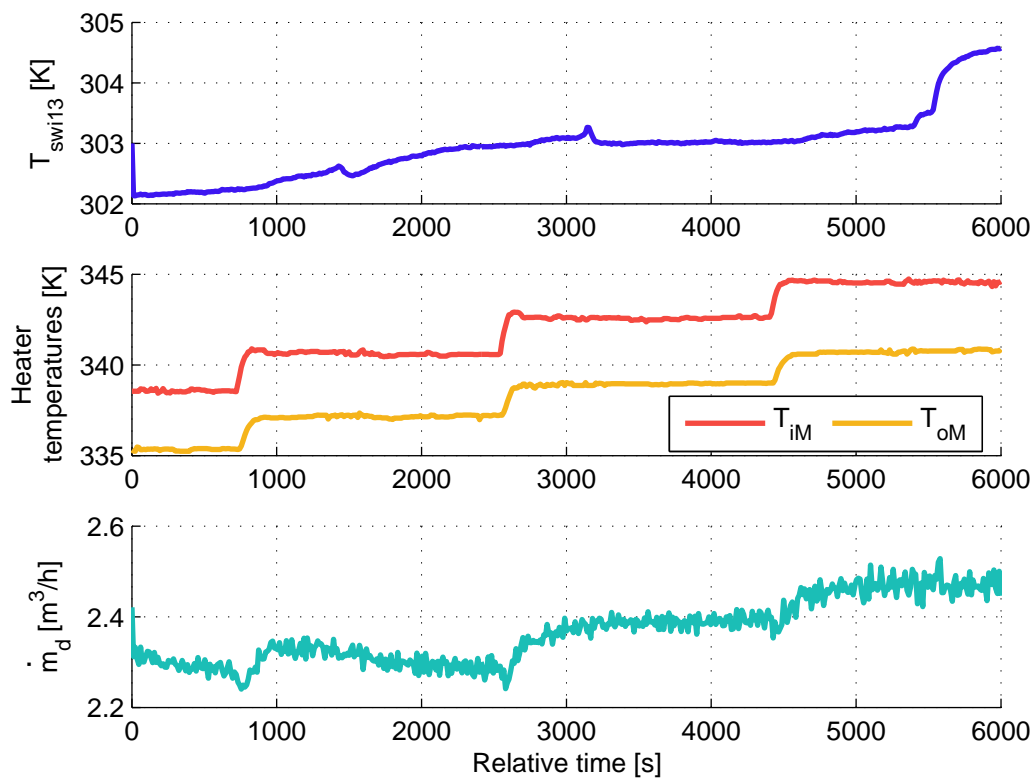


Figure 6: Simulation results of the MED unit model

- [10] E. Mathioulakis, V. Belessiotis, and E. Delyannis. Desalination by using alternative energy: Review and state-of-the-art. *Desalination*, 203:346–365, 2006.
- [11] B. Milow and E. Zarza. Advanced med solar desalination plants. configurations, costs, future—seven years of experience at the plataforma solar de almeria (spain). *Desalination*, 108(1-3):51–58, 1997.
- [12] P. Palenzuela, D. Alarcon, J. Blanco, E. Guillen, M. Ibarra, and G. Zaragoza. Modeling of the heat transfer of a solar multi-effect distillation plant at the plataforma solar de almeria. *Desalination and water treatment*, 31(1-3):257–268, 2011.
- [13] İ.H. Yılmaz and M.S. Söylemez. Design and computer simulation on multi-effect evaporation seawater desalination system using hybrid renewable energy sources in turkey. *Desalination*, 2012.

Modeling a drum motor for illustrating wearout phenomena

Olaf Enge-Rosenblatt¹, Christian Bayer¹, Joachim Schnüttgen²

¹Fraunhofer Institute for Integrated Circuits, Division Design Automation,
Zeunerstraße 38, 01069 Dresden, Germany

{Christian.Bayer; Olaf.Enger}@eas.iis.fraunhofer.de

²Interroll Holding GmbH, Interroll Research Center,
Lothforster Straße 32-40, 41849 Wassenberg, Germany
J.Schnuettgen@interroll.com

Abstract

In this contribution, a model of a drum motor is presented. This model was designed for description of dynamic behaviour of the drum motor as well as for the possible implementation of several wearing phenomena. Using this model, a better understanding of wear and tear phenomena has been achieved by carrying out a considerable number of simulation runs using different operational and wearing conditions. Using this information, important knowledge about detection of wearout signs was able to be gained.

Often, mathematical models with different levels of detail are used. In these cases, it may be a difficult task to obtain reliable parameters. In this paper, we present three different approaches for establishing a model structure and for the determination of needed parameters. This way, we were able to define every part of the model with an appropriate level of detail and equip them with adequate parameter values.

Keywords: drum motor; mathematical model; wearout phenomena modeling; parameter determination; condition monitoring

1 Introduction

Applications of mathematical models of technical systems are widespread in today's product development cycle. Mathematical models help to increase the understanding of physical properties of a product. The usage of mathematical models in the design phase allows investigations of functional properties under changing operational conditions. Both properties and operational conditions are described in the models by certain parameters. In the early phase of product development, only a certain range of values for these parameters is needed. Later on, these pa-

rameters have to be determined with higher accuracy to benefit from the model-based investigations.

Correct and robust operation under changing circumstances is the most important requirement concerning machines and facilities in today's industry. Additionally, all equipment must guarantee a very high level of availability. These two demands are competing against each other because every technical system is characterized by a certain appearance of wear and tear. This applies to mechanical and electrical systems but also for any other physical domain. This appearance of wear and tear increasingly causes a less correct operation of any technical system with progressing time of operation. Therefore, compliance checks concerning the allowed tolerances have to be performed either in certain time intervals or depending on the current condition of wearing. However, those checks take time and, therefore, decrease the machine's availability.

Using a mathematical model of a machine or a facility that is able to reconstruct phenomena of wearing is one promising way of getting out of the dilemma. Still, the model must be able to describe functional and dynamic properties, too. That is the reason why such mathematical models cannot be implemented in an easy and straight forward manner. The model structure developed first has to be laid out with necessary parameters. Some of them can be calculated while other ones may only be measured. Calculation may be performed analytically or, e.g., by using a Finite Element model. Parameter measurements mostly need considerable effort for establishing an appropriate test set-up. All three methods were applied for the development of the model presented here. Using a well-parameterized model, we can carry out investigations about impacts of effects of wear and tear on functional properties of a machine or a facility.

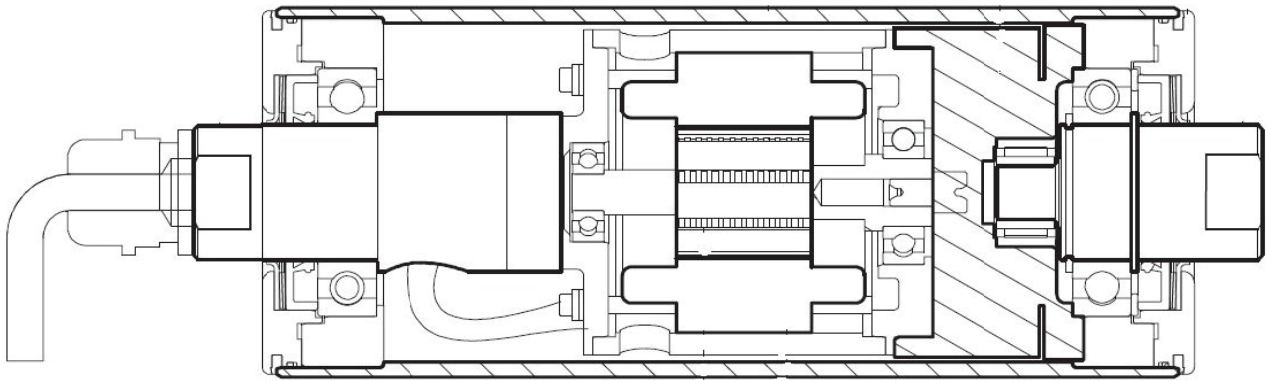


Figure 1 Sectional view of drum motor

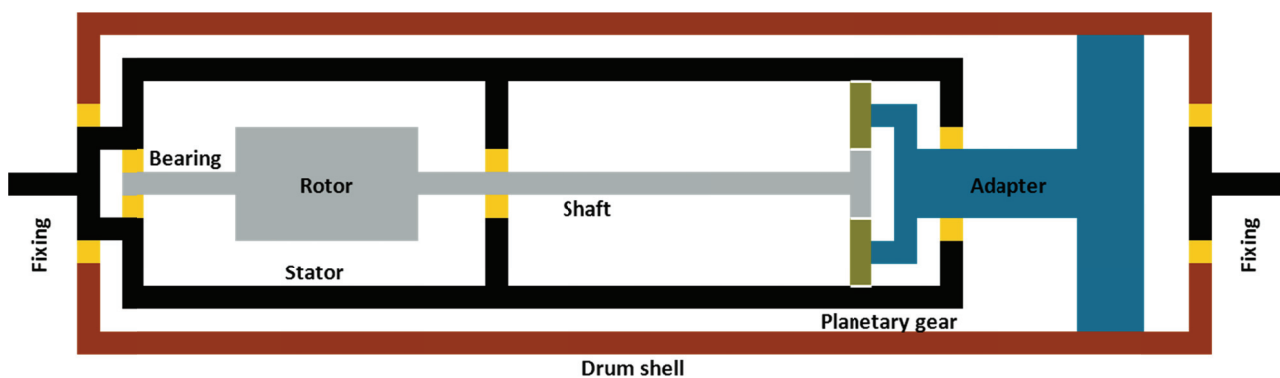


Figure 2 Sketch of the components of the drum motor

In this paper, a mathematical model of a drum motor is presented. This model regards important wearout phenomena occurring within the drum motor's components. Important model parameters are determined by analytical investigations, by Finite Element methods or by extensive measurements. Some parameters are included into the model by look-up tables. Finally, some simulation results performed to prove the functional properties of the model are given.

2 Design of a drum motor

A drum motor (or sometime referred to as a motorised pulley) is a component for driving a pulley for conveyor belts. A sectional view of such a motor is depicted in Figure 1. Figure 2 shows a sketch of the main components belonging to a typical drum motor. From both figures, one can understand the operation of the drum motor. The stator of the electric motor and the other black parts (including the ring wheel of the planetary gear) are fixed at the outer space. The grey part consists of the rotor of the electric motor

and the shaft connected to the input of the planetary gear (sun wheel). Both are driven by the motor's electromagnetic force. The carrier of the planetary gear is shown in blue colour (Adapter) while the planetary wheels are filled light green. All yellow parts stand for roller bearings. The two outer bearings are the main bearings of the drum motor. They have to resist the forces caused by the belt and its load. The three inner bearings have to guide the rotor of the electric motor and the gear's carrier. Finally, the drum shell is depicted in red colour. It is tightly attached to the adapter and, therefore, driven by the planetary gear carrier.

This mechanical system was modeled using Modelica because this multi-physical language is well-prepared for implementation of different approaches for parameter definitions [4]. The Modelica model diagram is shown in Figure 3. The connection to a model of the electrical subsystem of the electric motor (including controlling algorithms and power inverter) is realised by one input signal (torque) and two output signals (angle and angular velocity). The electrical subsystem's model is provided by a third party ([5]) and will not be discussed here in detail.

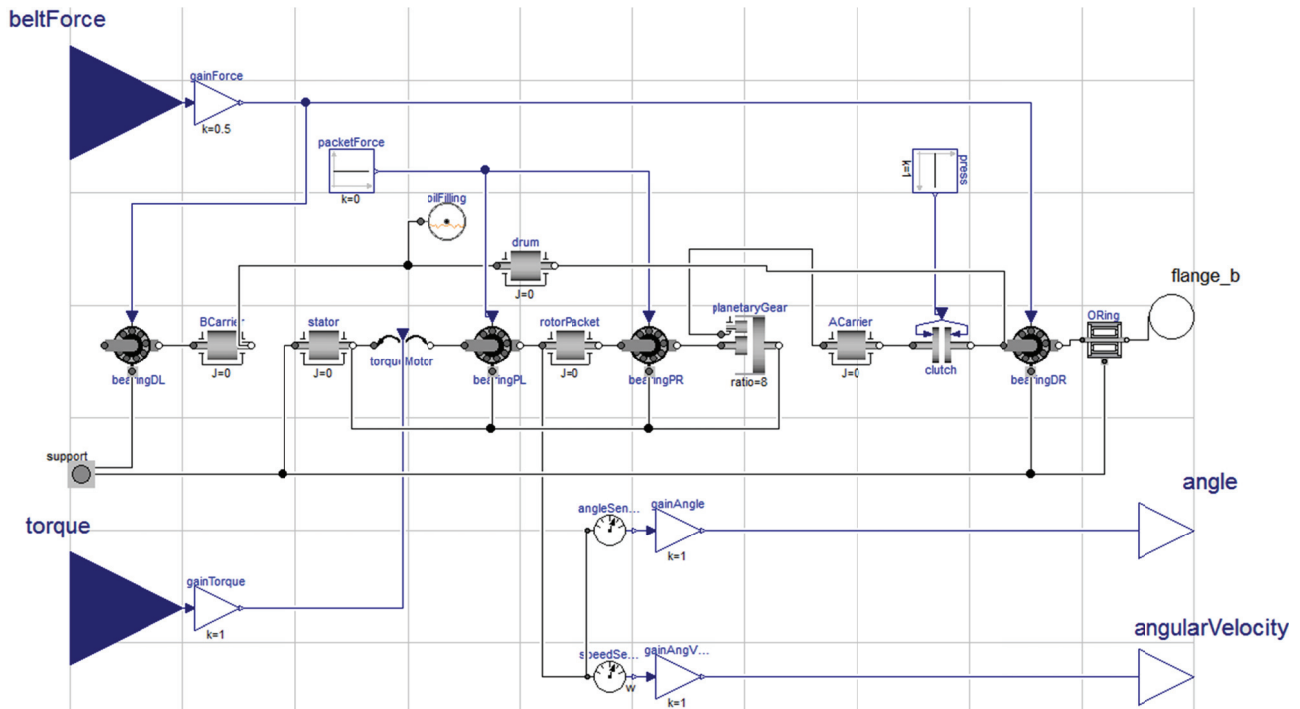


Figure 3 Modelica model diagram of the drum motor

3 Modelica model

The Modelica model (see Figure 3) consists of some main modules each describing a mechanical subsystem. First, there are the main roller bearings (“bearingDL” and “bearingDR”) between the drum shell (“drum”) and the fixed part (“support”) as well as the smaller roller bearings (“bearingPL” and “bearingPR”) between the rotor of the electric drive (“rotorPacket”) and the fixed stator (“stator”) or the planetary gear carrier (“ACarrier”), respectively. Second, the friction because of the O-rings is taken into account (“ORing”) and, third, a gear is included into the model (“planetaryGear”). The additional modules (e.g. “oilFilling”) are not of importance here.

All of the used sub-models can be supplemented by several wearout phenomena. The model’s level of detail defines which phenomena can be included. The more details are implemented in a model the more phenomena can mathematically be described. Some effects can be formulated by investigating physical relations, other effects, however, have to be described by general behavioural models which only give an approximated representation of static and dynamic properties regardless of real physical reasons behind them.

In the following, we present the most important parts of our model of a drum motor.

3.1 Roller bearings

The model of the roller bearings is implemented as a cylindrical joint component with only one rotational degree of freedom. Hence, it is designed much simpler than e.g. in [1] or [11]. The model is realized as an extension of the BearingFriction model of the Modelica Standard Library [9]. That means that both rotational axes of inner and outer ring are identical. The friction definition of the Standard Library has been complemented by friction effects caused by the rolling elements (balls in many cases), which are placed between the inner and outer ring of the roller bearing (see e.g. [12]). The model is used for belt drive applications, where an external force can act on the bearing. Hence, the BearingFriction model was extended by a RealInput interface to connect any external force value applied to the bearing. Typically, the direction of this force is fixed and thus the force is absorbed by all balls located on one half-space of the bearing.

Figure 4 shows a simple sketch of a roller bearing. It illustrates the application and the distribution of forces. The belt force F_B is divided into several sub-forces F_B^i . The resulting additional friction torque depends on the mechanical strain of the balls and their slight deformation.

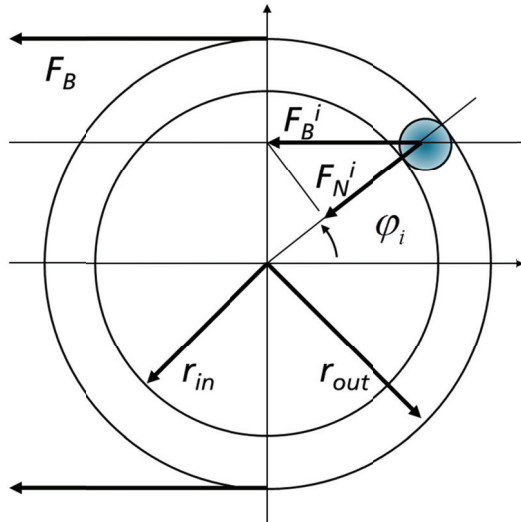


Figure 4 Sketch of a roller bearing

The pressure of each ball can be described by the Hertzian contact stress ([6], [7], [10]), which is basically modeled by a non-linear spring system. The displacement value for this model originates from a very small relative shift between the axes of inner and outer ring when a force is applied. Hence, the displacement value can be calculated analytically for each ball with respect to its angular position φ_i . We found that it is a good approximation to consider the displacement δ_i to be proportional to the term $\cos\varphi_i$. According to the Hertzian contact model the normal force to a ball then becomes

$$F_N^i = \tilde{F}_B (\cos\varphi_i)^{3/2}, \quad (1)$$

where \tilde{F}_B is a force value which will be determined by the external force applied to the bearing. Figure 4 shows that the normal force F_N^i is a projection of the partial external force F_B^i to the radial direction. F_B^i can therefore be expressed as

$$F_B^i = \frac{F_N^i}{\cos\varphi_i} = \tilde{F}_B \sqrt{\cos\varphi_i}. \quad (2)$$

From equation (2), the distribution of the external belt force according to angular ball positions is given. Note that only balls of the respective half-space are considered as the remaining balls are not affected by the force. Finally, the force value \tilde{F}_B is determined by the fact that all sub-forces F_B^i sum up to the belt force F_B and thus reads

$$\tilde{F}_B = \frac{F_B}{\sum_i \sqrt{\cos\varphi_i}}, \quad -\frac{\pi}{2} \leq \varphi_i \leq \frac{\pi}{2}. \quad (3)$$

The force value \tilde{F}_B is actually time-dependent. The force acting on one ball changes not only with angular position but also with the number of balls present in the half-space. Eventually, this causes slight fluctuations of the torque needed to operate the bearing continuously.

One of the models purpose is to simulate abrasion, which can be treated as an increased rolling friction F_R^i . The friction force is proportional to the known normal force F_N^i and thus depends further on a factor k_R^i for each ring, which might be a function of the angular position φ_i measured relative to the ring under consideration. We defined these factors by two look-up tables for all angular positions between 0 and 2π . This introduces high flexibility for the definition of damaged spots on the rings. The absolute value of the friction torque due to rolling balls then reads

$$\tau_R = \sum_i F_N^i (r_{in} k_{R,in}^i + r_{out} k_{R,out}^i). \quad (4)$$

where r_{in} and r_{out} are the radii of inner and outer ring. For abrasion simulation it is necessary to calculate the angular ball positions relatively to the ring positions, which are known from the connecting flanges. Moreover, the rotation speed of the balls differs from the speed of inner and outer ring and the rings might both rotate. However, they are a suitable reference system and the relative positions become

$$\varphi_i^{in} = -\frac{\varphi_{in} - \varphi_{out}}{1 + \frac{r_{in}}{r_{out}}} + \varphi_{i,0} \quad (5a)$$

$$\varphi_i^{out} = \frac{\varphi_{in} - \varphi_{out}}{1 + \frac{r_{out}}{r_{in}}} + \varphi_{i,0}. \quad (5b)$$

Equations (5) describe the angular position of a ball with respect to the inner and outer ring, respectively, where $\varphi_{i,0}$ denotes an initial position of the ball. Using (5), we can implement wearout phenomena depending on relative angles between rolling elements and both rings. This way, the most important forms of damage of roller bearings (e.g. dull surface, surface depressions like pittings) can be modeled by a variable friction depending on momentary angles of inner and outer ring.

3.2 O-ring friction

The drum motor is partly filled with oil which must not leak out during operation. At least two so-called O-rings are used for sealing both ends of the drum shell preventing the oil from leaking out. But they introduce high friction values and, therefore, they are of very high influence on dynamic behaviour of the drum motor.

To model such friction, extensive measurements were carried out by Interroll as will be described in Section 4. The goal of these investigations was not only to account for the non-linear behaviour but also to obtain useful absolute values for the friction torque. We found that the friction values depend mainly on the material of the O-rings as well as on the used lubricant (used or used not, lubricant grade, lubricant amount). But the friction behaviour for a fixed combination of material and lubricant shows a simpler structure in comparison with the friction behaviour of roller bearings. A first approximation of the friction may be a combination of a velocity-proportional share and a nearly constant share. Hence, it can be written like

$$\tau_o = k_o \omega + \tau_{o,0}, \quad (6)$$

where ω is the rotational speed of the drum shell. The parameters k_o and $\tau_{o,0}$ have to be found by performing appropriate experiments. A second approximation was implemented by full correlation of torque with rotational speed by using a look-up table of measured values. To this end, the standard library model BearingFriction was finally used to model the O-rings because it offers such possibilities.

For both approximations, we assumed that O-ring friction does not vary with any outer load. The belt forces and any load in axial direction are completely absorbed by the main bearings. Hence, wearout phenomena at O-rings are expected to be uniform, i.e. they are not dependent on the drum shell's angle. But we found that O-ring friction vary both with hours of operation – because of wearing in effects – and with increasing wearout, e.g. by abrasion. These influences have to be determined by extensive measurements, too.

3.3 One-stage gear

The drum motor includes a single-stage or multi-stage planetary gear, which can be implemented by the standard library model IdealPlanetary or a combination of two of it. However, it does not include any features to emulate wearout effects. As a first approach the usually existing backlash and rotational

stiffness can be inserted between the driving shaft and the gear's input wheel by a rotational spring damper system. The backlash influences the dynamic behaviour of the drum motor and is a measure for abrasion.

Gears have special kinds of tooth flanks to minimize abrasion [2], [13]. Most gears use the evolvent tooth flank, which ensures that there is no grinding between teeth and that the transmitted torque has no ripple. During operation the flank might degrade due to high loads or improper maintenance. Even teeth may break off under certain circumstances. As it would be very challenging to model these effects through multi-body simulation, we developed a Finite Element (FE) model starting from a two-wheel gear. In the ideal case, both wheels change their angular position simultaneously but with opposite direction. Actually, the driven wheel moves first and yields a slight deformation of teeth according to the Hertzian contact model. This effect is shown in Figure 5 for one wheel. The torque applied to the driven wheel is transmitted by this means. The deformation leads to a little deviation $\Delta\varphi$ in angular position of both wheels and depends also on the absolute position φ of the wheels, as the contact point moves along the evolvent. The transmitted torque thus is a function of $\Delta\varphi$ and φ which is determined by the FE simulation.

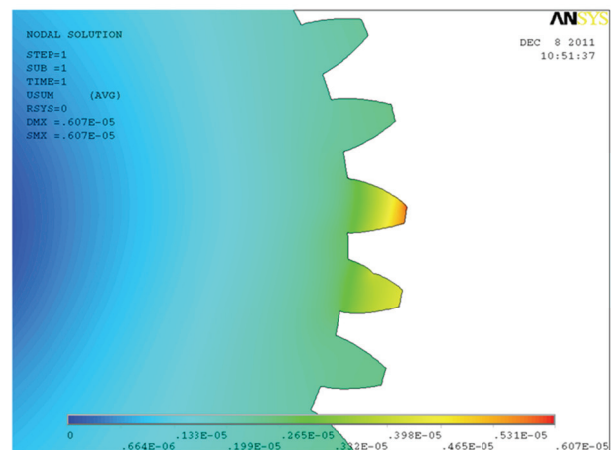


Figure 5 Gearwheel with stretching of teeth (Note that the deformation is displayed using a magnification of 100 in relation to the shape of the wheel)

Figure 6 shows the resulting torque function for two gear wheels of equal size. We can deduce that for small loads, i.e. for small $\Delta\varphi$, the transmission is independent of the angular position φ and depends linearly on $\Delta\varphi$. However, for heavy loads the

transmission function becomes non-linear and also depends on φ . Note that the torque function is periodic in φ .

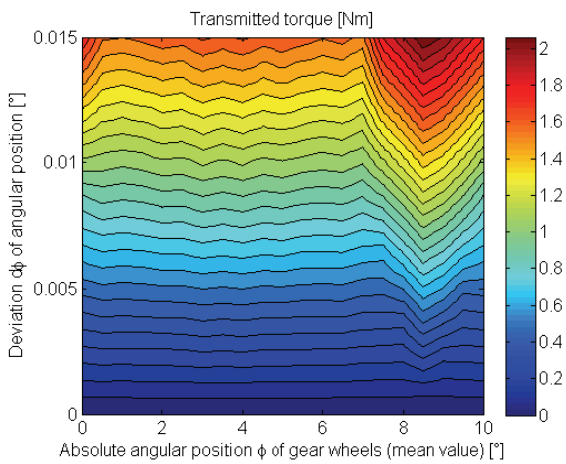


Figure 6 Dependency of torque on both angles and their difference

The results of the FE simulation can be used to develop a Modelica model of any gear wheel combination. The investigation of different tooth flanks and their abrasion is thus possible and even the effect of missing teeth can be evaluated.

3.4 Environment of the model

The drum motor is intended for belt drive systems. Such environments can be modeled by the commercial Belts library, which includes elements for the static and dynamic analysis of belt drive systems. The key components are belt spans and pulleys. We added interfaces to our drum motor model such that it can act as a pulley and is fully compatible to the library components. This way we are able to simulate the motor within a realistic environment and to investigate the effects of different signs of wear and tear.

Our Modelica model does not include the electric motor. This component was provided by a third party as a Matlab Simulink model [5]. It provides the driving torque and uses angular position and velocity from our model to control the electric motor. The co-simulation was carried out using the Dymola block feature in Simulink (see [8]). The purpose of this combined model is to investigate the influence of mechanical properties and setups to the electronic parts and controller of the motor. But these investigations are not content of this paper.

4 Determination of parameters

Determination of parameters used within the model is a very important task. Some parameters can be read from geometrical values like masses and moments of inertia. But other parameters like friction coefficients need to be determined by specific measurements carried out using real devices.

For an appropriate assignment of parameters needed to measure, a number of roll-out tests were performed. These tests were focused on the determination of friction losses because of bearings and O-rings without any influence from electric motor or gear. To this end, a single return pulley was accelerated to a certain rotational speed. After that, the pulley is retarded only by friction torques in bearings (first test set-up) or by friction torques in bearings and due to friction between drum shell and O-rings (second test set-up).

Temperature and lubrication are additional physical quantities influencing the friction torques. But temperature can be ignored if all measurements are carried out within a range of temperature like in a typical use case. Lubrication, if effected correctly, was found to be of no noticeable influence.

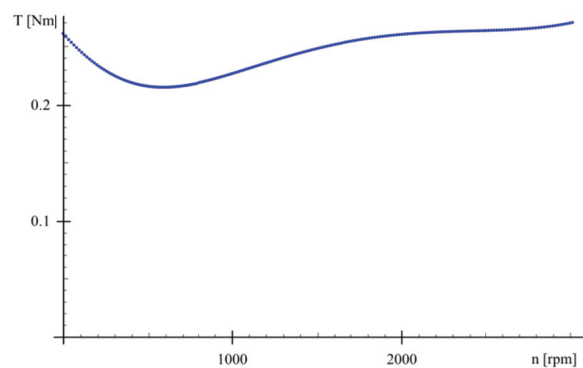


Figure 7 Friction torque without any O-ring

Figure 7 shows the curve of friction torque measured across rotational speed if no O-rings are attached to the test set-up. It is a result of retarding friction because of bearings only. The curve shows a shape which is nearly independent on rotational speed because torque values range from 0.2 to 0.3 Nm (see also [12]). In a first approximation the friction torque is, therefore, implemented in the model by a constant value of Coulomb friction. A better integration of the measured curve can be performed using a look-up table containing momentary friction torque values depending on rotational speed.

Figure 8 depicts the measurement result if both outer O-rings are attached to the drum shell. This is a

result of retarding friction because of bearings and O-rings. The curve shows a non-linear dependency on rotational speed. For simplification, the curve is split into one part with constant friction (approximately 0.95 Nm) and one part with a constant coefficient for friction depending linearly on rotational speed. The constant value is about 0.1 Nm/1000 rpm. Here again, a better integration of the measured curve can be performed via a look-up table.

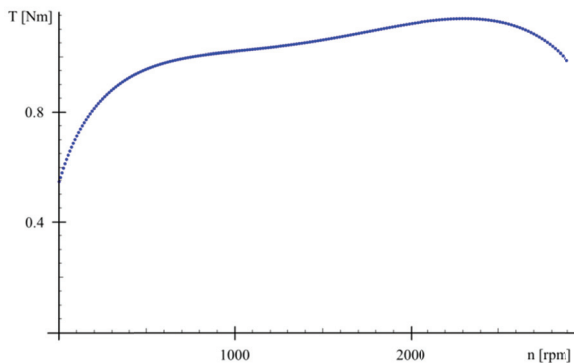


Figure 8 Friction torque with O-rings

5 Simulation results

As an example, we present simulation results regarding the roller bearings. Because of lack of measurement results up to now, we can unfortunately only present simulation results. These results were obtained using Dymola [3]. The application environment of the drum motor consists of a load by a constant transversal force (like force F_B in Figure 4). This force is induced by the belt and reaches up to 1 kN in a typical application.

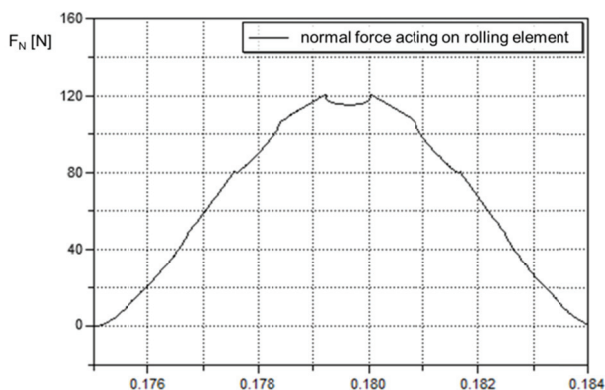


Figure 9 Normal force against time acting on one ball located within the force loaded half-space of the roller bearing

First we investigated the normal force acting on the balls of a bearing. This gives an impression how the bearing model works and where the torque ripples originate from. Figure 9 shows the result across time for the motion of one ball within the force-loaded half space of the bearing (which is located in the half circle on the right hand side in Figure 4). The normal force is expected to be zero within the other half space (left hand side in Figure 4). Furthermore, we assume that there are no slip effects between ball and inner or outer ring of the bearing. Hence, the same curve shape results when depicting the normal force across the angular position of the ball instead of time. The friction torque for the one considered ball can then be determined from the normal force like shown in Section 3.1. This way, the ball-specific friction value can be calculated at every point in time depending on the ball's position and the bearing's transversal load force. Applying this result for every ball of the bearing, the complete friction can be calculated from present bearing angle and the load force.

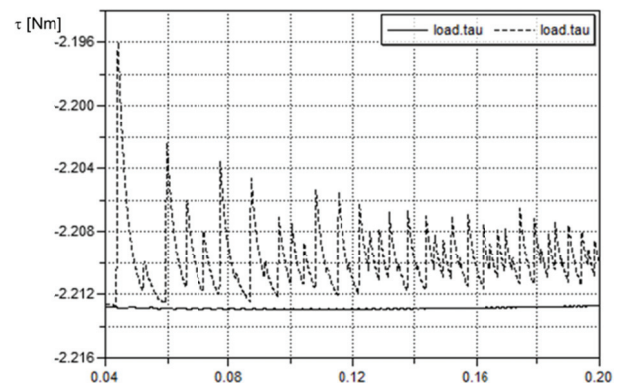


Figure 10 Torque transmitted to the load displayed across time without damage (solid line) and with localized damage on the outer ring (dashed line)

To investigate the resulting torque ripple at a rotating load (not to be mistaken for the transversal load force), the drum motor was driven by a constant torque and the rotational load was emulated by a linear and velocity dependent damper. To show the effect of a spot damage at one of the rings, such a damage was introduced by a localized increase of the friction coefficient k_R . This increase was calculated dynamically using the present bearing angle. We compared the torques acting on the load with and without spot damage, respectively. Figure 10 shows the results. Both torques are depicted across time for a certain range after the transient effect has been vanished. The torque without bearing damage (solid

line) is nearly constant during the complete simulation interval. Contrastingly, the torque with bearing damage (dashed line) shows distinct ripples. The ripples' magnitude order is about 0.5 % of the torque's mean value. That seems to be not so high but it is enough to detect some differences within a frequency plot. This way, a damage of sufficient dimension could be detected by a dedicated condition monitoring system.

6 Conclusions

We presented an approach to simulate wear and tear phenomena within complex systems. For the case of a drum motor we proposed three different methods of modeling and parametrising components thereof. The roller bearing was described analytically, whereas the gear was simulated using the Finite Element method. A third access to unknown parameters is measurement as shown with the O-rings. Using these well-parameterised models, we were able to establish a behavioural description for some important wearout effects with the drum motor. Hence, these models can be used to predict the behaviour of a worn system within its usual environment. This opens the possibility to investigate some consequences of wearout effects in several simulation runs in order to establish design rules for condition monitoring algorithms and thus support the development of adapted condition monitoring systems. This in turn allows for improved maintenance strategies and reduced costs.

Acknowledgement

This work was funded by the German Federal Ministry of Economy within the project AutASS.

References

- [1] Ashtekar, A.; Sadeghi, F.; Stacke, L.-E.: Surface defects effects on bearing dynamics. Proceedings of the Institution of Mechanical Engineers, Part J: Journal of Engineering Tribology, Vol. 224, No. 1, pp. 25-35, 2010
- [2] Colbourne, J.R.: The geometric design of internal gear pairs, AGMA Technical Paper, 87 FTM 2, 1987
- [3] Dymola 7.3, Dassault Systèmes
- [4] Fritzson, P.: Principles of Object-Oriented Modeling and Simulation with Modelica 2.1, IEEE Press, 2004
- [5] Herold, T.; Franck, D.; Lange, E.; Hameyer, K.: Extension of a d-q model of a permanent magnet excited synchronous machine by including saturation, cross-coupling and slotting effects. IEEE International Electric Machines Drives Conference (IEMDC), Niagara Falls, Ontario, Canada, 15-18 May, 2011, Proc. pp. 1363-1367
- [6] Hertz, H.: Über die Berührung fester elastischer Körper. Journal für die reine und angewandte Mathematik, 92:156-171, 1881
- [7] Hertz, H.: Über die Berührung fester elastischer Körper (On the contact of rigid elastic solids). In: Miscellaneous Papers. Jones and Schott, Editors, J. reine und angewandte Mathematik 92, Macmillan, London, pp. 156ff., 1896
- [8] MATLAB® 2010a, The MathWorks, Inc.
- [9] www.modelica.org/libraries/Modelica/
- [10] Paland, E.G.: Technisches Taschenbuch. INA Schaeffer KG, 2002
- [11] Stacke, L.-E.; Fritzson, D.: Dynamic behaviour of rolling bearings: simulations and experiments. Proceedings of the Institution of Mechanical Engineers, Part J: Journal of Engineering Tribology, Vol. 215, No. 6, pp. 499-508, 2001
- [12] Tan, X.; Modafe, A.; Ghodssi, R.: Measurement and modeling of dynamic rolling friction in linear microball bearings. Journal of Dynamic Systems, Measurement and Control, Transactions of the ASME, 128 (4) , pp. 891-898, 2006
- [13] Zhuravlev, G.A.; Ageev, A.I.: Gear Teeth Bending Strength Analysis. Izvestia vyssih ucebnyh zavedenij. Masinostroenie (11), pp. 44-48, 1978

“Green Building” – Modelling renewable building energy systems and electric mobility concepts using Modelica

René Unger¹, Torsten Schwan², B. Mikoleit¹, Bernard Bäker², Christian Kehrer³, Tobias Rodemann⁴

¹EA EnergieArchitektur GmbH, Dresden, Germany

²Institute of Automotive Technologies Dresden - IAD, Dresden University of Technology
Dresden, Germany

³ITI GmbH, Dresden, Germany

⁴Honda Research Institute Europe, Offenbach/Main, Germany
rene.unger@ea-gmbh.de, schwan@iad.tu-dresden.de, kehrer@iti.de
Tobias.Rodemann@honda-ri.de

Abstract

For most people, a comfortable living and mobility are basic needs. With the rising individual demand for energy as well as the diminishing fossil energy resources, new optimized concepts for energy supply and usage are required. To address these challenges, renewable energy sources, decentralized storage, and electric mobility concepts are matters of rapidly growing importance.

Future building energy systems have to successfully integrate user demands, local renewable energy, storage systems and charging infrastructure, a task requiring extensive scrutinizing.

Typical questions to the engineer are to compare different system layouts with respect to sustainability, cost, and robustness, or to identify the right levers in an energy system to optimize components and control algorithms.

This paper describes an approach to solve such questions using simulations with the non-causal language Modelica. Modeling paradigms and examples are shown. Special emphasize is given to the “Green Building” library and its components, bringing major building energy systems and electric vehicles to the same platform.

Keywords: Renewable energy; Building; eMobility

1 Introduction

Increasing the use of renewable energy for almost all aspects of people’s life is one of the major topics of this decade. Energy storage, smarter energy consumption and interaction of energy grid components, on global scale as well as locally, are tasks to be solved by the engineer.

Ecological footprint, detail efficiency as well as usage comfort are matters becoming more important. To fulfill these aspects, various components like photovoltaic or storage tanks, even weather forecast, need to work together to satisfy the users’ demands in a renewable and reliable way.

In addition to the technical aspects of automation and networking systems, the functionality of this component interaction needs to be clarified. Heat, electricity and mobility used to be separated aspects in life. With the use of renewables these are increasingly correlated. For example a combination of micro-wind-turbines, photovoltaics, solar heat and heat pumps could be used in a specific building. Another possible solution would be a combined heat and power unit (CHP) heating the house and charging the electric vehicle. The solution may also vary depending on the available monetary budget.

Renewable energy is limited in availability. The peak PV-power is at noon while peak consumption is often in the morning or in the evening. The need for energy storage or at least time shifting of consumption arises. Even user behavior is important in such a system.

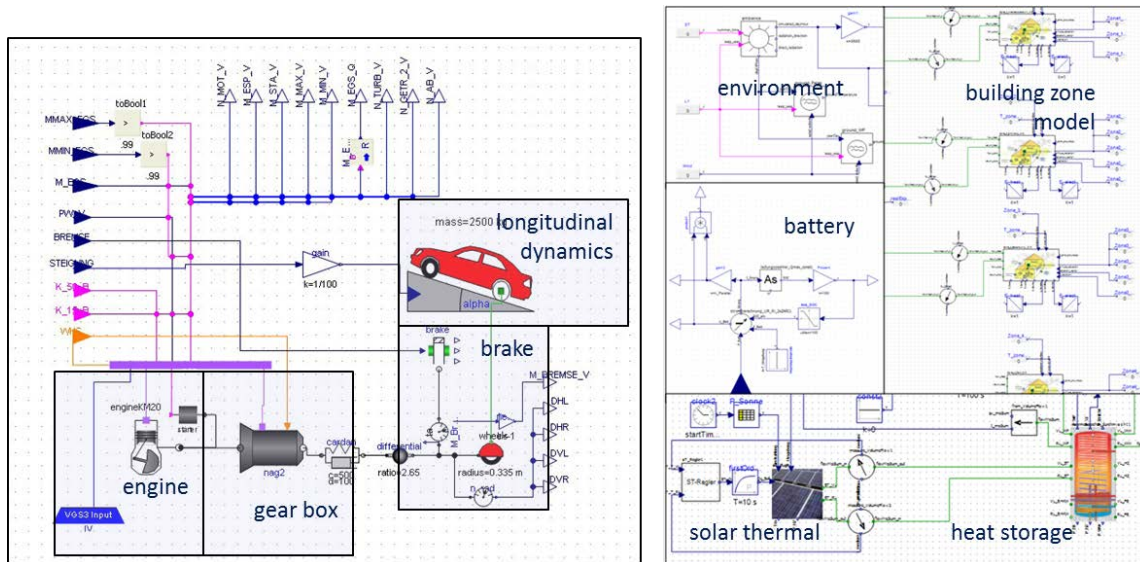


Fig. 1: Modelica-based simulation models for vehicles (left) and buildings (right) [1, 5]

Hence, to find a suitable energy system configuration for a specific scenario has become an extensive engineering task. Therefore, new supporting tools and methods covering the whole system are needed.

2 Simulation Tools

Today’s available tools can be categorized into different groups. First, there are special component layout programs like for example PVSol for photovoltaics.

A second group of tools uses FEM and CFD. These make it possible to simulate heat and radiation input to complex rooms and buildings and to calculate the resulting temperature fields, air flow, etc. EnergyPlus, ANSYS and Ecotect are examples for these powerful tools.

A third group addresses systems simulation. HVAC, even photovoltaics and wind are integrated into one block oriented system model. The underlying physics are often represented as equivalent networks while control algorithms are represented in a signal oriented way or are programmed in a procedural language. A typical tool-chain would contain TRNSYS and Matlab Simulink. These toolchains are extraordinary powerful. Yet some important effects like the nondeterministic behavior of humans, electric mobility, dynamic costs, battery aging and probability-based energy management systems have been difficult to implement.

Modelica, as a non-causal, non-proprietary and cross-domain modeling language with Tools like SimulationX excel in these requirements. Some Libraries like Modelica “Buildings” or “Human Comfort” contain models for building energy and com-

bined control system simulations [8]. Yet, with these libraries, it is still a huge effort to model a complete building-vehicle-user energy system.

In such a holistic simulation, systems of fast dynamics (1ms-1s) like vehicle physics or battery control have to work alongside with systems of low dynamics (1min-1day) for a long simulation time span (days to years).

Additionally the models need to be complex enough to test control algorithms but not too complex for a fast simulation speed as well as safe and easy to configure. A systems engineer as addressee of the simulation is specialized in component interaction, not in heat pump specifics, vehicle batteries and detailed building thermodynamics. For special cases where detail is needed, simulator coupling is an option.

To fulfill these requirements, an approach widely used in the automotive industry was adapted to the field of building energy system modeling and realized in the Green Building Simulation Library.

3 Modelica Green Building Library - modeling paradigms

The Green Building Library was created using the cross-domain equation based concept of the Modelica language. The aim was to create a set of physical and functional models with similar granularity and handling. This way a complete renewable energy system can be represented (Fig. 2), including:

- sources like photovoltaics, windmills, solar-thermal collectors, heat pumps or CHPs,
- storage tanks, batteries and grid,
- consumers, user behavior, weather as well as
- charging stations and electric vehicles.

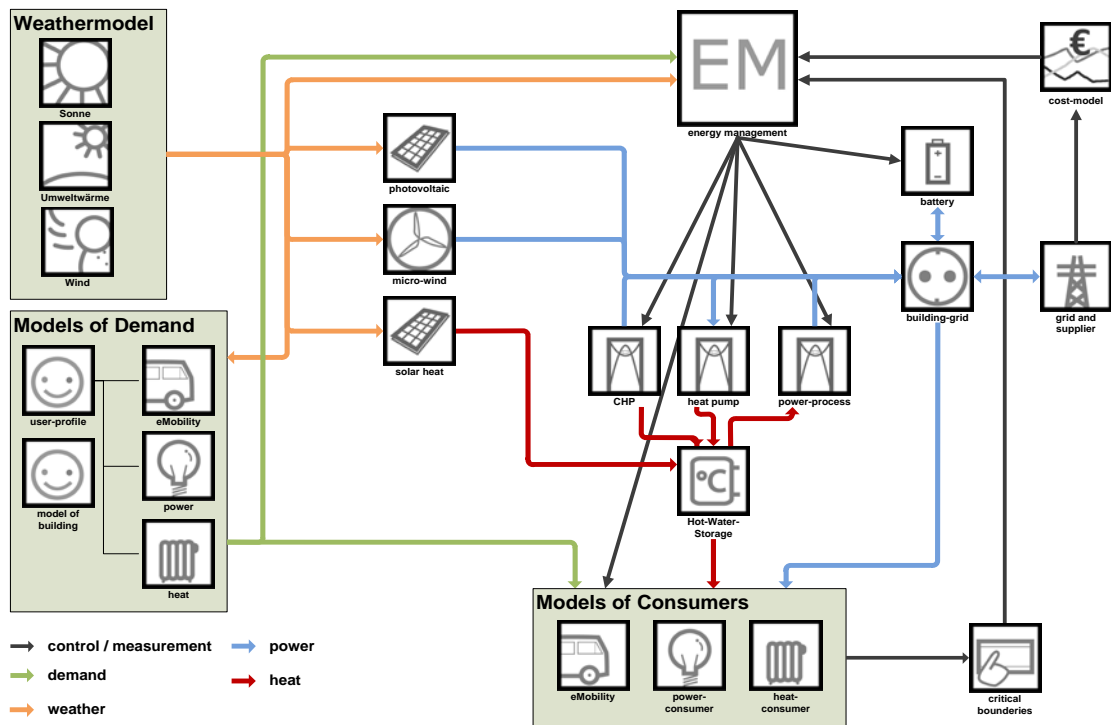


Fig. 2: Component models integrated in Modelica-library [2]

The granularity and complexity (fig. 1) of each element is comparable to the approach used in the automotive industry (e.g. partial models for engines, gearboxes and longitudinal dynamics of vehicles).

Special emphasis was placed on the input parameter set of each component and intuitive modeling. For better usability, all parameters are similar to those found in typical component data sheets or standardized reports (i.e. EnEV [10]).

Another requirement to the chosen approach is, that all the relevant characteristics needed for the comparison of different building energy systems, shall be calculated within one simulation environment, if possible. Therefore, all library components have been modeled as compatible differential-algebraic equation systems (DAE) including physical behavior, control algorithms and external interrupt connectors for energy management systems. Furthermore, optional functions to calculate investment and operating costs of each component can be used (fig. 3). Currently, however, these calculations are performed in external post processing routines.

Another important requirement is a high simulation speed while maintaining the highest possible time resolution. For advanced comparisons, a year needs to be simulated with a minimum step size of 1 minute; this is at least half a million steps for one simulation run. This is longer than the usual 15min – 1h time steps of thermal simulations. To achieve the needed model reduction fast internal processes were

simplified or replaced with functional descriptions, each as internal DAE systems.

A major factor for the simplification was the influence on outer processes and component interaction. A typical example is the windmill power electronics. Instead of simulated MOSFETs and capacitors, a phenomenological model containing characteristic curves for conversion factors, efficiency and voltage limiters represents the electronics.

In other models, where such simplification is no option (error margin, numeric stability), pre-calculation is used. Complex driving cycles are one example for fast internal time constants. Heat pump characteristics based on source and heating medium temperatures are another example, where the solver would need to calculate partial differentials.

Emphasize is given to an exact representation on effects which are relevant for the energetic behavior of a component (e.g. heat absorption and dissipation of a heat storage depending on the temperature spread). Other equations describing less relevant effects, like volume flow within heat storages, are neglected. This constant granularity is a usability advantage over usage of models from different sources.

Influences like weather are integrated using an ambience block, which reads either external databases or location provided with the library as internal blocks (fig. 3).

Most of the library components require special control algorithms for the internal regulation of the

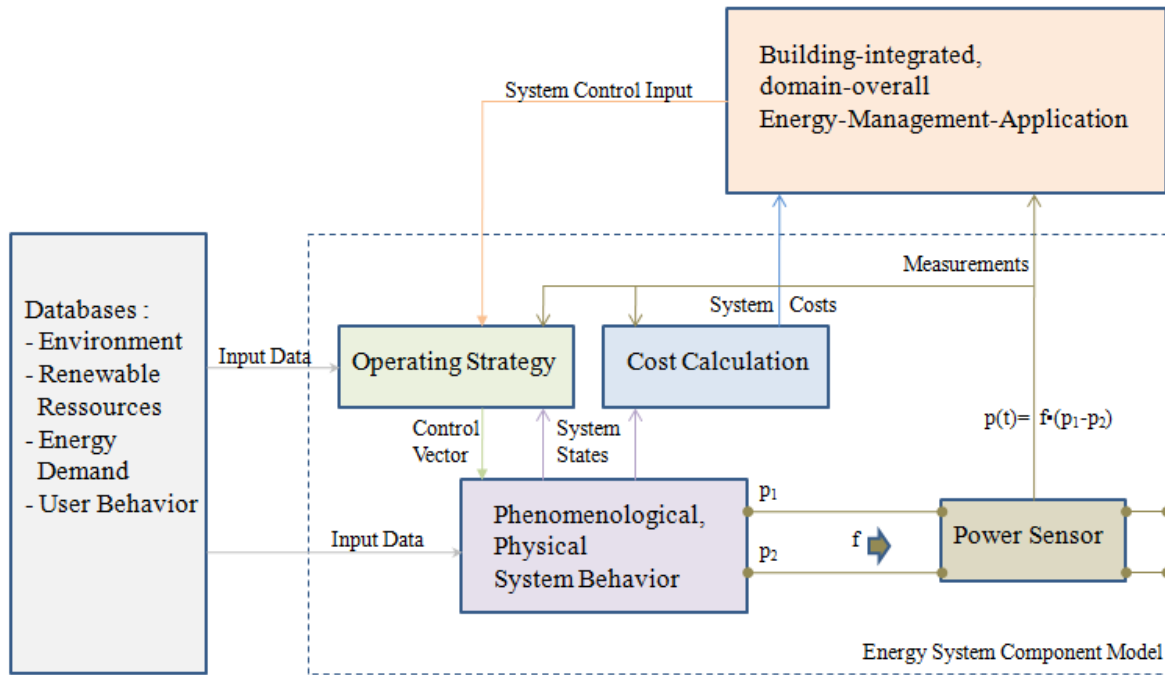


Fig. 3: Generalized component layout [1]

system states (e.g. de-icing processes of air heat pumps). These control strategies are integrated as blocks in the general component layout to simulate the typical behavior of the components. The user, to test new operating strategies like a combined heat/power-led CHP usage depending on vehicle dock and weather forecast, can replace the control block. Another option is the use of controller inputs for a superimposed energy management (fig. 3).

Within a domain, the components are connected with specific connector elements, which are derived from “real-world”-connections like pipes for heating systems and cables for electrical systems. This way, the real energy exchange is simulated and can be observed during and after each simulation run [1].

$$e_{therm}(t) = \int \rho_{med} \cdot c_{p_{med}} \cdot \dot{V}_{med} \cdot \Delta T_{med} dt \quad (1)$$

$$e_{el}(t) = \int U_{eff} \cdot I_{eff} \cdot \cos \varphi \cdot n_{phase} dt \quad (2)$$

Both consistent equations (1) and (2) describe the interchanged energy between connected components. Therefore, both equations consist of domain-specific flow (volume flow and current) and potential (temperature spread and voltage) states as well as further special constant values.

4 Coupling of Building and Vehicle

As explained before, the main challenge in coupling building and vehicle is the difference in the major time constants, which would lead to a small minimum simulation step size.

The thermal energy flow in buildings mainly depends on low system dynamics caused by inner masses, slow external temperature changes, etc. Electrical demand changes faster but it still is in the order of minutes at the building’s lateral.

In contrast to that, vehicle-specific time constants are much lower, in the order of milliseconds to seconds and they are vital for an adequate simulation of the energetic behavior (fuel, electrical energy).

There are proven frameworks to dynamically simulate the energetic behavior of vehicles with alternative drive trains [see for example 9]. These contain approaches to simulate the driver’s behavior and the operating strategy defining the vehicle operation mode (EV mode, ICE mode) as well as the detailed dynamic behavior of each vehicle component (e.g. gear boxes, ICE, EM etc.).

Unfortunately, such approaches are too complex for implementation in the combined simulation of buildings and vehicles. Because of the differing system dynamics, direct coupling of building and vehicle creates very stiff DAE systems. Hence, either the simulation time becomes non-acceptable or the results deteriorate in accuracy and numerical stability.

To solve this we looked at the vehicle from the building point of view. This way the vehicle is a component, which is either docked at home or executing a certain driving cycle. If it is available and connected to a charging station (battery electric or plug-in hybrid) then the vehicle is a consumer or an intelligent battery. The important information is the energy or fuel needed for a specific cycle, vehicle availability and if the vehicle is connected, the power

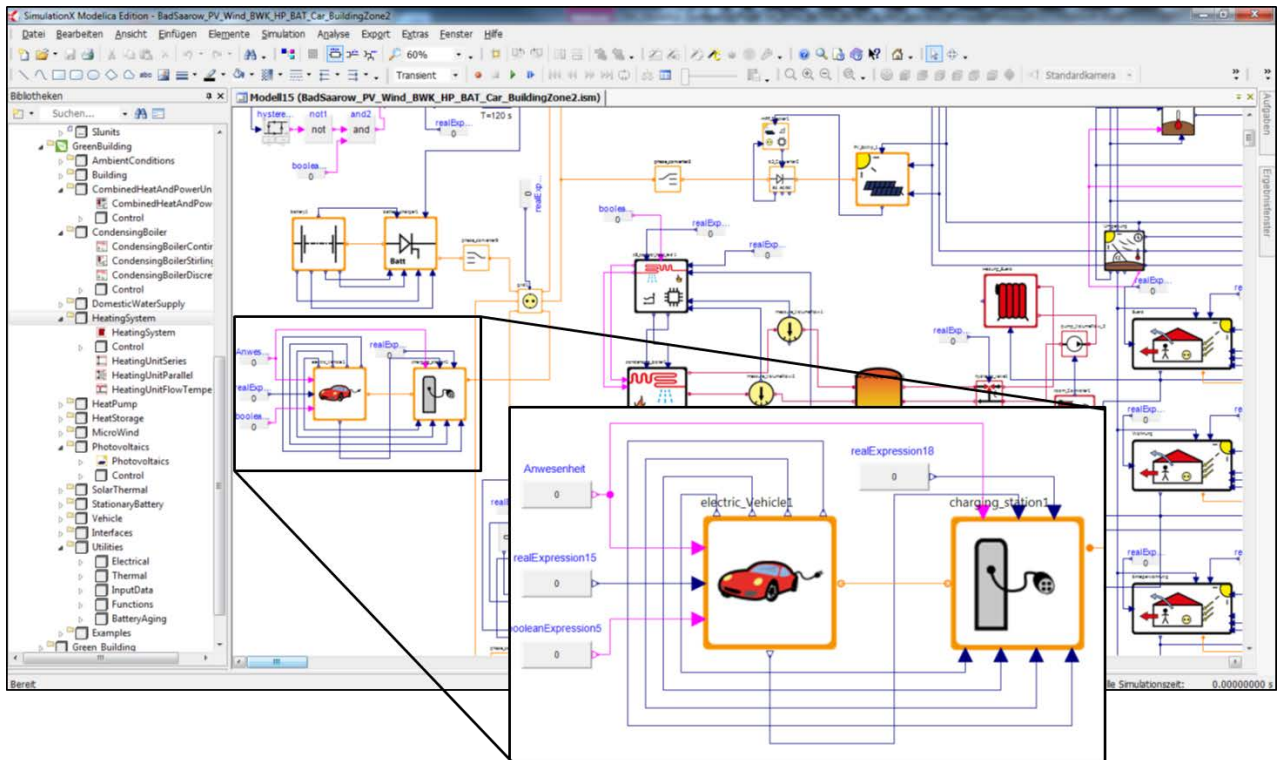


Fig. 4: Vehicle and charging station as a part of a complete energy system modeled in SimulationX

exchange and control algorithms. The energy demand during the driving cycle does not directly interact with the building. Therefore, this can be represented by pre-calculated driving cycles, user models and the simplification of the vehicle to a battery model.

This way in the building simulation, the vehicle is considered as only one component of the building energy system (fig. 4) with focuses on charging and grid-support. In a preprocessing operation, the driving cycles for different vehicle internals (ICE, BEV, PHEV, etc.) are simulated with high accuracy. This includes detailed longitudinal dynamics, architecture (serial, parallel etc.), power train components characteristics (e.g. battery size) and selected operating strategy (e.g. deplete and sustain) [see 9 et al.]. The main results are mean speed, mean power and fuel or energy consumption for use within the building energy simulation.

The vehicle presence can be either derived as a transient simulation variable from the driving time in the preprocessing or created as synthetic user behavior for the presence at the charging station.

Using that approach allows a high accuracy of the fuel and electrical power demand of the vehicle during a time period with an adequate simulation speed. Furthermore, the feedback of the charging strategy and the power supply to the vehicle on the building

energy system can be analyzed. Influences of superior energy management systems on the electrical energy supply to the vehicle and the amount of renewable energy used for driving the vehicle are analyzable as well.

5 Exemplary simulation results

An example, showing capabilities and power of the presented “Green Building”-library is a semidetached house in Germany. One simulation represents a conventional energy system and ICE vehicle. A second simulation shows a complex renewable configuration.

The conventional and the renewable configuration are then assessed regarding:

- primary energy balance,
- carbon dioxide emission balance and
- renewable fraction of traction energy

For these assessments of annual balances, a statistical method was used. Analogue to VDI 4655, twelve reference days were defined (i.e. sunny winterday, weekend). Each reference day was simulated. The results were superimposed with the weather statistics of the last 10 years.

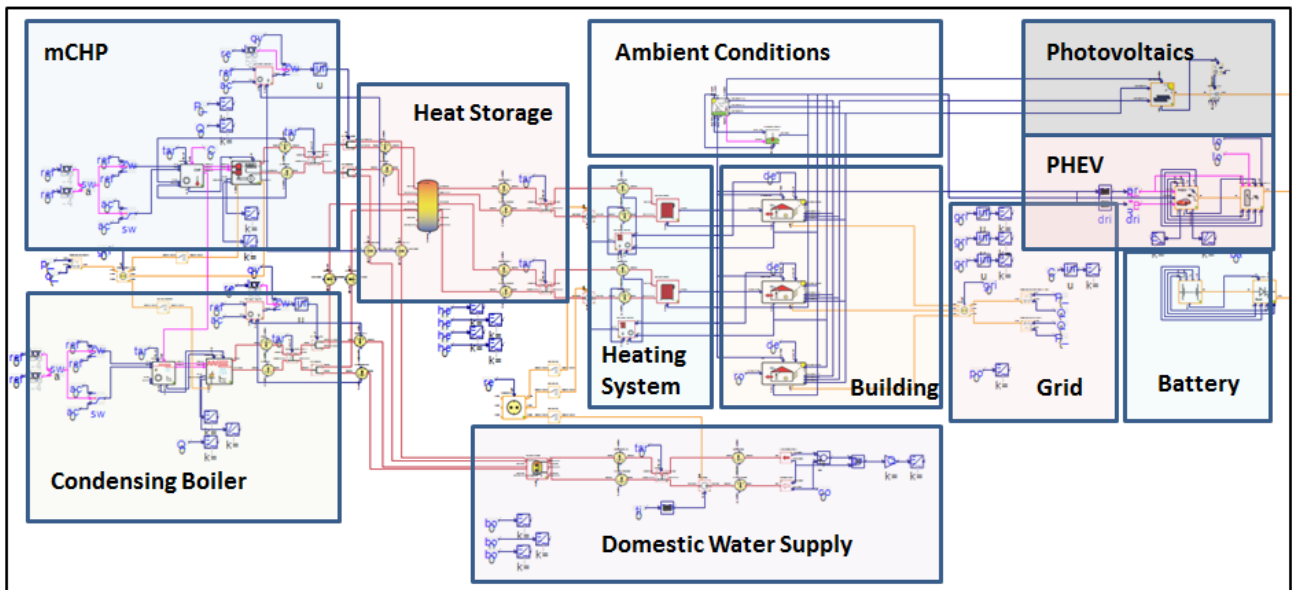


Fig. 5: Renewable energy system model

The renewable energy system consists of (fig. 5):

- small combined heat and power unit
- gas-fired condensing boiler
- heat storage
- domestic water boiler
- stationary battery
- photovoltaic system
- plug-in-hybrid-electric-vehicle

In opposite to that, the conventional energy system only contains three components (fig. 6):

- gas-fired condensing boiler
- domestic water boiler
- compact car (ICE)

Both systems were simulated in combination with a 3-zonal building model with floor heating system. Component parameters (e.g. battery size) and system control algorithms were adapted to the requirements

of the exemplary building scenario. However, parameter variations (e.g. variation of battery size) could easily be done due to the flexibility of the library components.

Although, there are many other analyzable criteria, for renewable energy systems, the ecological footprint is one of the most important bases of the decision-making. It is possible to evaluate this footprint by using primary energy factors. These factors bias different forms of energy (e.g. electricity, fuel, natural gas) by describing how much primary energy (e.g. coal-equivalent) is required for their provision.

Fig. 7 shows the primary energy balance of the analyzed system variants. The renewable system has a better primary energy balance and ecological footprint than the conventional system since a big share of the electrical energy is generated by CHP and photovoltaics.

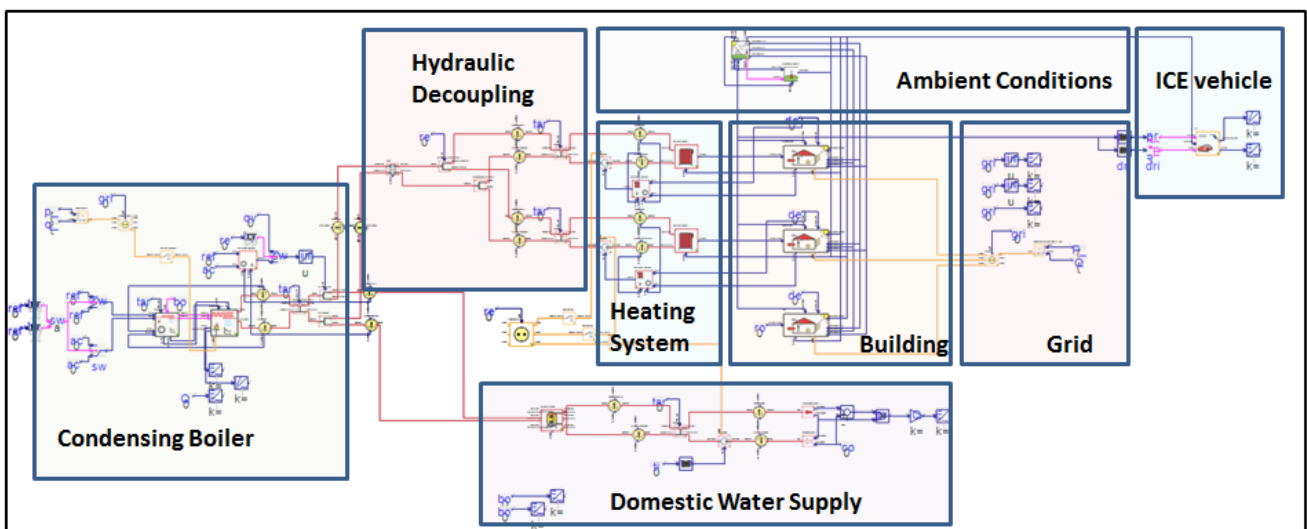


Fig. 6: Conventional energy system model

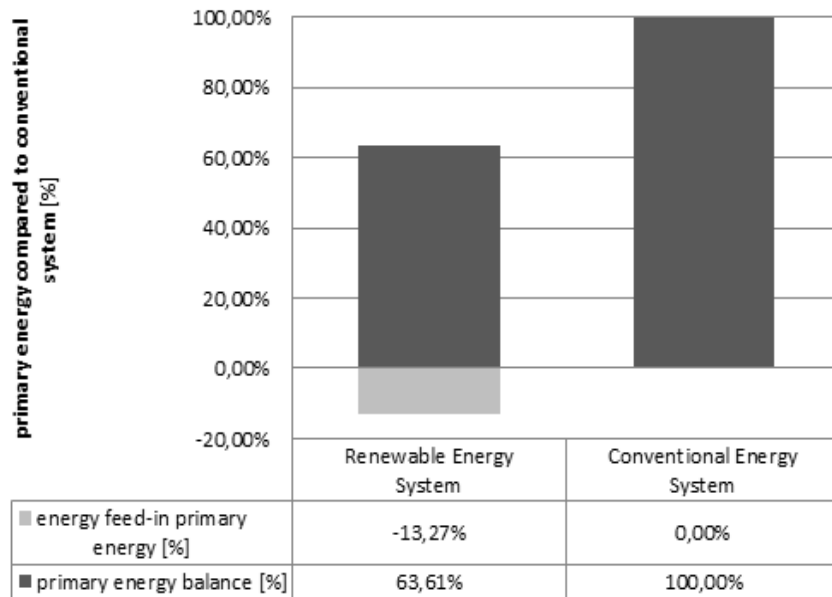


Fig. 7: Comparison of relative primary energy balance

Another important criterion for evaluation of different building energy systems and connected vehicles with (partly) electrified power train is the annual carbon dioxide emission balance. In this process all emissions of vehicles, heating system and electrical energy consumption are analyzed.

Fig. 8 shows that the annual CO₂ emission and electrical energy consumption of the renewable system with PHEV are much lower than the ones of conventional system. The main reason is the electricity output of photovoltaics and CHP. This helps to

maintain a high renewable mileage of the PHEV in a typical commuting situation with 30 km of daily driving. With the regarded vehicle, this cycle could be covered almost completely on battery. The only exceptions were some winter days where the combustion engine was needed.

Compared to the conventional system, the CO₂ emission for heating in the renewable system is higher. This is caused by electricity conversion and by the marginally lower thermal efficiency of the CHP (92% of CHP compared to 98% of condensing

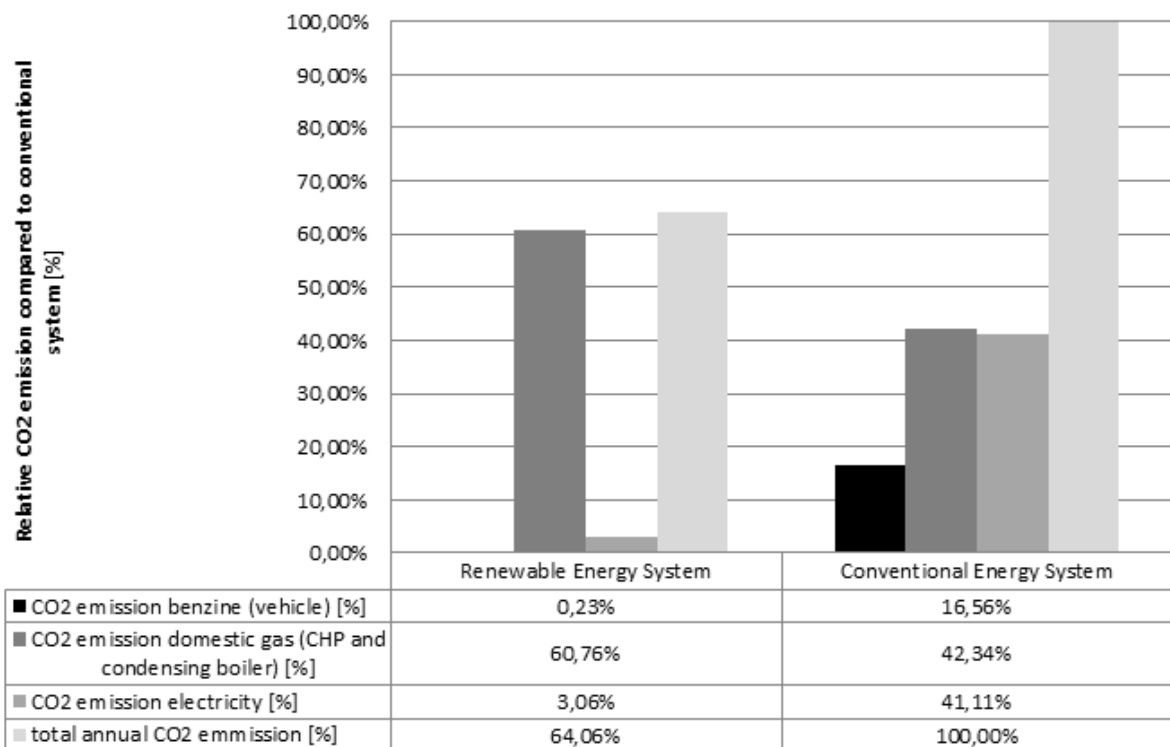


Fig. 8: Comparison of relative primary energy balance

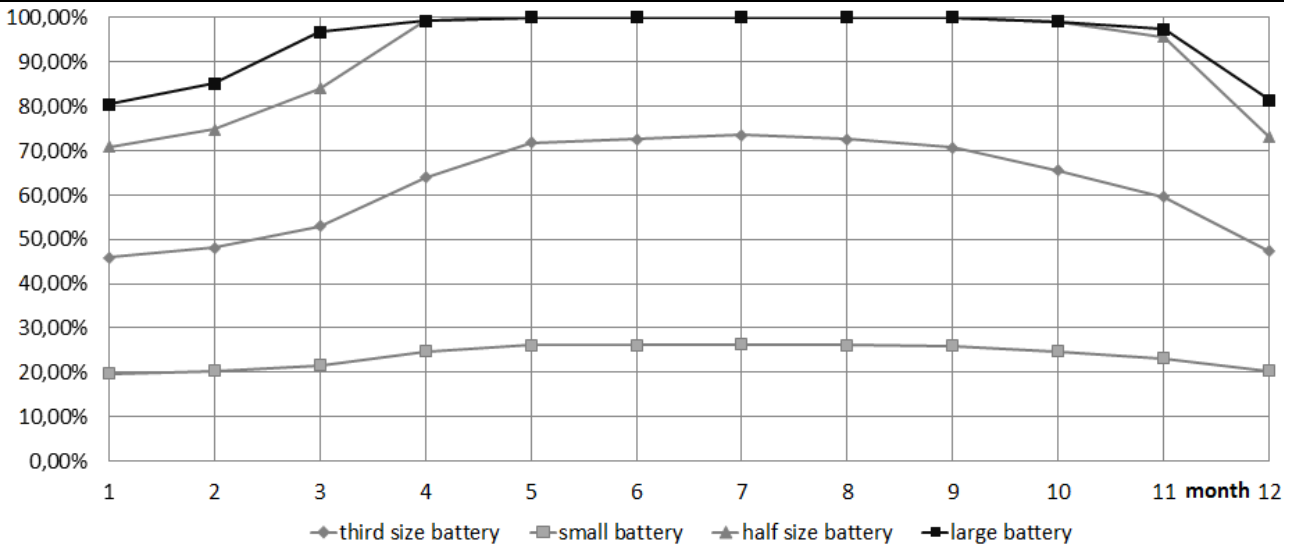


Fig. 9: Ratio of renewable energy to total vehicle energy demand

boiler).

The total CO₂ emission of the whole energy system (building combined with vehicle) of the conventional system is about 55% higher than the one of the renewable system. This significant impact of the system on the environment can now be measured against investment cost or production resources.

The system can be optimized further to avoid grid storage (grid feed-in) or towards a maximum renewable vehicle mileage.

For the ratio between renewable energy and fossil fuels used to fulfill the individual mobility demand, i.e. the annual distance driven only using renewable energy, the size of the stationary battery is essential.

With the stationary battery, the energy income (CHP, PV at noon) can be decoupled from energy usage (charging at night). The basic simulated operational strategy (also defined in Modelica) was to charge the stationary battery whenever photovoltaic

energy was available and to transfer this energy to the vehicle when docked. The difference to the energy requested by the vehicle was taken from CHP and grid. Values for losses in battery and converters matched those of the real components. This way a bigger amount of PV-energy in the stationary battery results in a higher renewable mileage.

To evaluate the influence of stationary battery capacity, the scenario was simulated with four sizes of relative 100%, 50%, 30% and 10%. Fig. 9 shows that the PHEV with the biggest battery size is driven with an average ratio of 95% renewable energy during the year while the smaller ones have ratios of 91%, 62% and 24%.

So, with a slightly worse primary energy balance (about 7%) due to less grid-feeding, the biggest battery offers almost complete renewable mileage for the PHEV. With the half size battery, the coverage is still more than 90%.

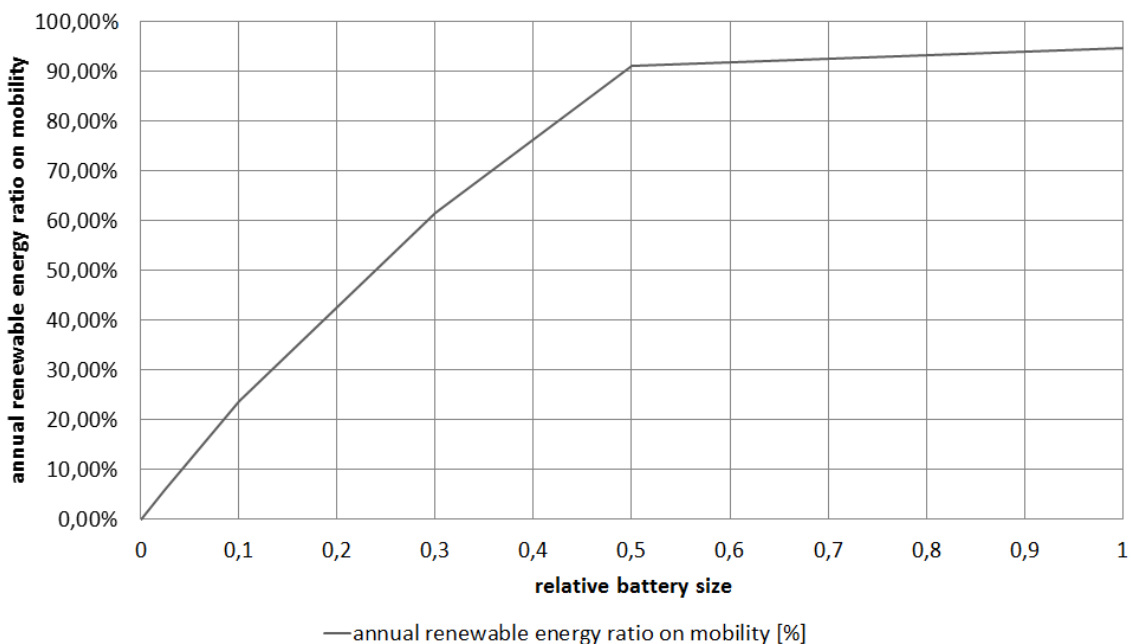


Fig. 10: annual renewable ratio of vehicle energy depending on relative battery capacity

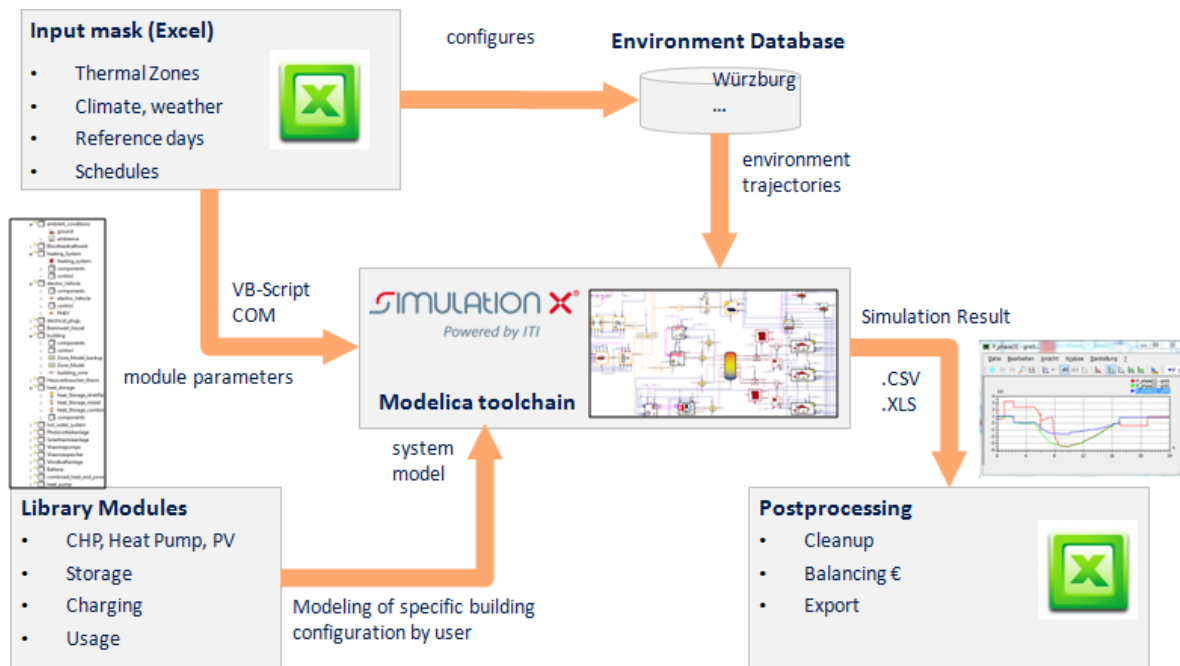


Fig. 11: Toolchain of simulation framework [1]

Obviously, with increasing battery size, i.e. capacity, the achievable annual renewable energy ratio on mobility increases monotonically (fig. 10). Because of the presented operating strategy for the photovoltaic system, the more capacity of the stationary battery is available the more renewable energy can be used to recharge the vehicle. An asymptotic maximum (fig. 10) occurs because the PHEV uses the internal combustion engine under cold outside temperatures, thus not using renewables.

Comparing battery costs leads to an optimum regarding battery size and annual renewable mobility energy ratio. This optimum can be calculated using annual capital costs for battery versus the achievable renewable energy ratio on annually driven distance.

The same evaluation and optimization could be done using different operational strategies, component (PV) sizes, vehicles or driving cycles. Even robustness to stochastic user behavior could be analyzed using the described holistic energy simulation approach.

6 Summary and Conclusions

The presented Modelica-based simulation library enables the modeling of various architectures for building energy systems including vehicles with (partly) electrified power trains. Simulation with these models creates a multitude of results, which can be used for evaluating and optimizing these systems using different criteria. Some criteria like battery size were presented within this paper. The more complex

energy systems get, the larger the potential for optimization.

Besides the evaluation of energy system variants, the new Green Building framework (fig. 11) offers the capability of model-based development also for energy management algorithms in buildings or predictive renewable operation strategies for vehicles with electrified power trains.

Current work aims to improve the library with new component models like phase changing material (PCM) thermal storages and more top-level models for even easier use. A second major research objective is to create new energy management algorithms for the complete system using real-prediction and simulation [3].

Since renewable energies are still expensive in terms of money and production resources, an efficient use of these systems is essential.

Acknowledgements

The library and methods described are developed within the research project “Residence and Mobility”. The research project is encouraged with subsidies from the European Union and the Sächsische Aufbaubank (SAB).

The simulation results presented in this paper arise from an exemplary research study, which was carried out in cooperation with Honda Research Institute Europe.



References

- [1] Schwan, T., Unger, R., Baeker, B., Mikoleit, B., Kehrer, C.: Optimization of local renewable energy systems using automotive simulation approaches. 12th Conference of International Building Performance Simulation Association, Sydney, 2011.
- [2] Schwan, T., Unger, R., Baeker, B., Mikoleit, B., Kehrer, C.: Optimization-Tool for local renewable energy usage in the connected system: Building-eMobility. 8th Modelica Conference. Dresden, 2011.
- [3] Unger, R., Schwan, T., Mikoleit, B., Baeker, B.: “Residence and Mobility” – Renewable Energy Management in the linked system “Building – eMobility”. 1st International Energy Efficient Vehicle Conference. Dresden, 2011.
- [4] Unger, R., Schwan, T., Mikoleit, B., Baeker, B.: Optimization-Tool for local renewable energy usage in the connected system: “Building-eMobility”. 13th ITI Symposium. Dresden, 2010.
- [5] Chrisofakis, E., Junghanns, A., Kehrer, C., Rink, A.: Simulation-based development of automotive control software with Modelica, 8th Modelica Conference, Dresden, 2011.
- [6] Fritzon, P.: Principles of Object-Oriented Modeling and Simulation with Modelica 2.1. Wiley-IEE Press, 2003.
- [7] SimulationX-Homepage of ITI GmbH. <http://www.simulationx.com>
- [8] Wetter, M., Zuo, W., Nouidui, T. S.: Recent Developments of the Modelica “Buildings” Library for Building Energy and Control Systems. 8th Modelica Conference. Dresden. 2011.
- [9] Kutter, S., Langhammer, S., Bäker, B.: eVehicleLib - Eine Modelica-Bibliothek zur Simulation von Fahrzeugen mit alternativen Antrieben; 20th Symposium Simulationstechnik - ASIM 2009. Cottbus. 2009.
- [10] Verordnung über energiesparenden Wärmeschutz und energiesparende Anlagentechnik bei Gebäuden - Energieeinsparverordnung, 2007.
- [11] PVSol Homepage of The Solar Design Company. <http://www.solardesign.co.uk/index.php>.
- [12] TRNSYS-Homepage of Transsolar. http://www.transsolar.com/software/docs/trnsys/trnsys_uebersicht_de.htm

High-Fidelity Transmission Simulation for Hardware-in-the-Loop Applications

Orang Vahid Paul Goossens

Maplesoft

615 Kumpf Drive, Waterloo, ON, Canada, N2V 1K8

ovahid@maplesoft.com pgoossens@maplesoft.com

Abstract

Model-based development plays a central part in optimizing existing transmission designs and exploring new system architectures. Design iterations and performance evaluations are done through virtual prototypes of the transmission systems, used in hardware-in-the-loop (HiL) simulations. In this paper, MapleSim's Driveline Component Library is introduced. The combination of this Modelica library and Maple's core symbolic technology, enables engineers to include more detail into their models targeted for real-time simulation of transmission systems. The paper also includes some results from the work at Aisin AW in modeling transmissions and HiL testing.

Keywords: Transmissions; Hardware-in-the-loop; symbolic calculations

1 Introduction

As automotive manufacturers strive to meet and exceed performance requirements on fuel efficiency and ride comfort, they have focused increasingly on the transmission design as one of the key factors. Engineers are putting tremendous effort into determining exactly how the power is lost, and what can be done to reduce losses and improve overall fuel efficiency. As a result, the transmission industry is now actively involved in optimizing existing transmission designs and exploring new system architectures. At the same time, transmission controllers are becoming more complicated and more detailed product testing is needed than ever before.

Model-based development (MBD) plays a central part towards achieving these goals. Design iterations are done through virtual prototypes of the transmission systems, used in hardware-in-the-loop (HiL) simulations. Virtual prototyping can yield more efficient products at significantly reduced costs by al-

lowing engineers to address design issues long before they invest in physical prototypes.

In this paper we report on some of the activities under taken at AISIN AW in Japan regarding HiL simulation and the use of MapleSim environment to accelerate the development of automatic transmissions. The requirements for low calculation cost plant models for real-time simulations were met by creating the gear train part of the model in MapleSim. These models are then exported as optimized c-code for implementation into the HiL system.

2 Transmission Modeling Using the Driveline Component Library

The transmission models referred to in this paper are built using the components from the MapleSim Driveline Component Library (DCL) [1] as well as other components from the Standard Modelica Libraries [2]. DCL covers all stages in a powertrain model from the engine through to the differential, wheels and road loads (See Figure 1). Furthermore, the library allows for flexible inclusion of power loss data that best reflect the way in which the loss data was acquired.

In the remainder of this section, some of the features of the components used in modeling transmissions are discussed.

2.1 Clutches and Brakes

As part of the standard component library, MapleSim provides two clutch models: a standard, controllable friction clutch and a one-way clutch [2].

In DCL, these models are expanded; clutch and brake models provide a real output port for the loss power and a Boolean output port to indicate clutch lock-up. There are also other formulation improvements that make DCL models perform better when used with fixed-step integrators usually encountered

in real time applications and Hardware-in-the-Loop simulations.

This is accommodated in the lookup table data by providing torque ratios and capacity values for

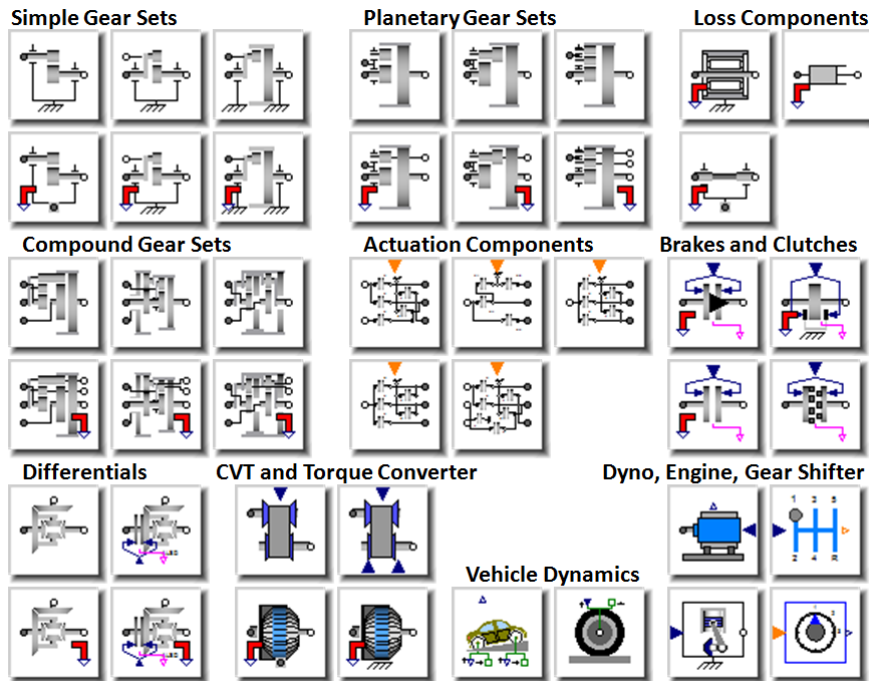


Figure 1: Driveline Component Library

2.2 Torque Converter

The torque converter is modeled using tables of measured data. The following characteristics are used:

- Torque Ratio $\frac{\tau_t}{\tau_p}$ vs Speed Ratio $\frac{\omega_t}{\omega_p}$
- Load Capacity C vs Speed Ratio $\frac{\omega_t}{\omega_p}$

Where subscripts “t” and “p” designate turbine and pump quantities, respectively. The required data can be given as tabulated data. The Torque Converter component supports two alternative formulations based on the following definitions of the load capacity :

- $C = \frac{\tau_p}{(\omega_p)^2} \left[\frac{\text{N.m}}{\text{RPM}^2} \right]$
- $C = \frac{\omega_p}{\sqrt{\tau_p}} \left[\frac{(\text{rad/s})^{1/2}}{\text{N.m}} \right]$

Backward flow, happens during deceleration of the vehicle where the vehicle kinetic energy is transmitted back through the transmission to the engine. In this situation, the turbine is pumping and the pump is acting as a turbine. Since torque converters are not designed to work optimally this way, the torque converter will have very different characteris-

$\frac{\omega_t}{\omega_p} > 1$, typically up to about 5.

In the test model shown in Figure 2, the input (pump) torque is increased linearly for the first 10 seconds. At low speeds, between $t = 0$ and 4 s, the turbine torque increases faster than the input torque. This is the “torque multiplication” effect typically seen in the torque converters [3]. Due to the inherent inefficiencies in the mechanism, the turbine speed is slightly less than the pump speed while the torque is driving the pump.

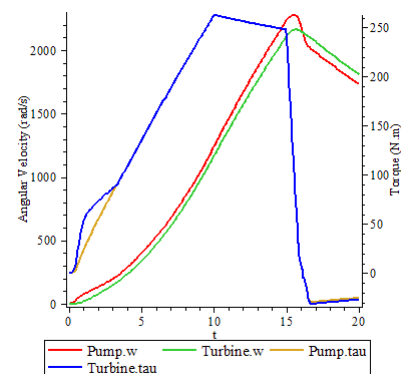
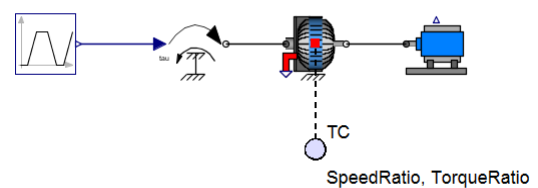


Figure 2: Torque converter test model

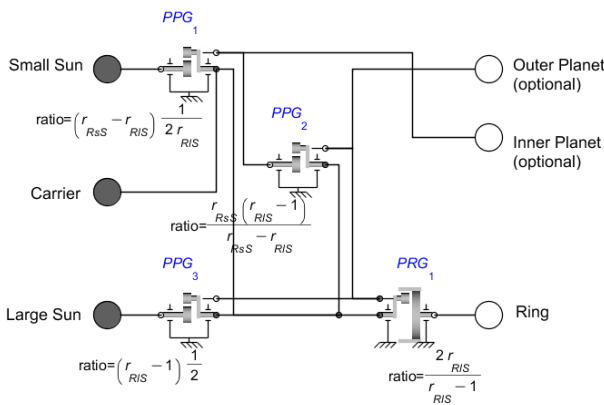
Note that when the input torque drops at $t = 15$ s, the kinetic energy of the dynamometer changes the torque flow from forward to backward (i.e. turbine drives the pump), and the pump speed drops below the turbine speed.

2.3 Gears, Gear Sets, and Transmissions

DCL includes simple and compound gear sets and related actuation components for modeling gear trains and transmissions (see Figure 1). The Ravigneaux gear set component is discussed in the following as an example of the compound gear components in DCL.

Ravigneaux Gear Set

The Ravigneaux configuration is a basic automatic transmission planetary assembly. As shown Figure 3, this configuration is constructed internally using three Planet-Planet components and one Planet-Ring component.



```
connect (PPG1.carrier, PRG1.carrier);
connect (PRG1.carrier, PPG2.carrier);
connect (PPG2.carrier, PPG3.carrier);
connect (carrier, PRG1.carrier);

connect (small_sun, PPG1.inplanet);
connect (large_sun, PPG3.inplanet);

connect (PPG1.outplanet, PPG2.inplanet);
connect (PPG2.outplanet, PRG1.planet);
connect (PPG2.outplanet, PPG3.outplanet);

connect (ring, PRG1.ring);

connect (inner_planet, PPG1.outplanet);
connect (outer_planet, PPG2.outplanet);
```

Figure 3: Internal structure of the Ravigneaux Gear

Lepelletier Gear Sets

There are two Lepelletier Actuation components (6-speed and 7-speed) provided in DCL which can be used together with a Ravigneaux gear and a planetary gear to create 6-speed or 7-speed transmissions as shown in Figure 4-a and 4-b, respectively.

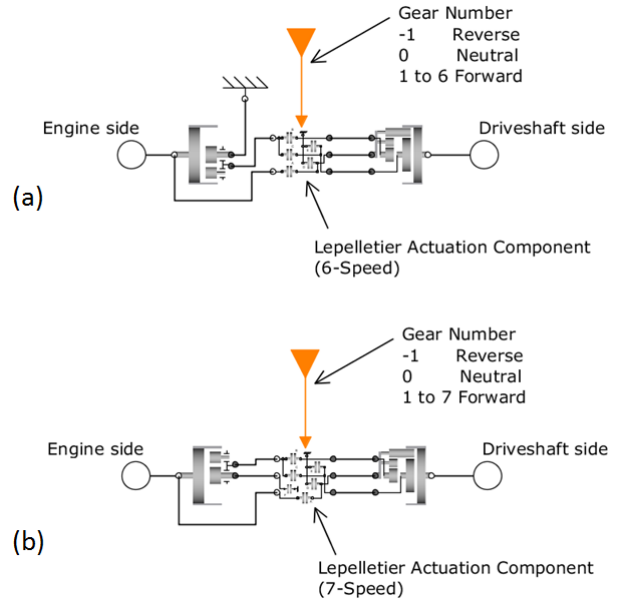


Figure 4: Building 6-speed (a) and 7-speed (b) transmissions with the Lepelletier Actuation components

2.4 Incorporating Losses

As shown in Figure 5, all of the gear components in the DCL can easily be switched from ideal (i.e. no losses) to lossy where power losses due to tooth-meshing are accounted for [4].

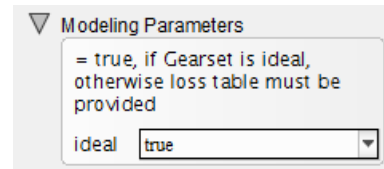


Figure 5: Fundamental GUI option for all gears – ideal = true/false

In lossy mode, the meshing friction is expressed as a transmission efficiency ($\eta(\omega) = \tau_{out}/\tau_{in}$) which may be defined as a function of the gear angular velocity via data tables. The user has the option to provide an efficiency table for each meshing gear pair in the gear set individually.

In compound gear sets (Planetary Gear, Dual-ratio Planetary Gear, Counter-rotating Planetary Gear, Ravigneaux Gear, Simpson Gear, and CR-CR Gear), internal bearing damping can be added using the component options. Bearing friction can also be added using external Bearing Friction components by enabling the optional “planet flanges”. Figure 6 shows a Counter-rotating Planetary Gear component with added bearing friction losses. The bearing friction is expressed as a torque loss and is related only to the shaft speed [2].

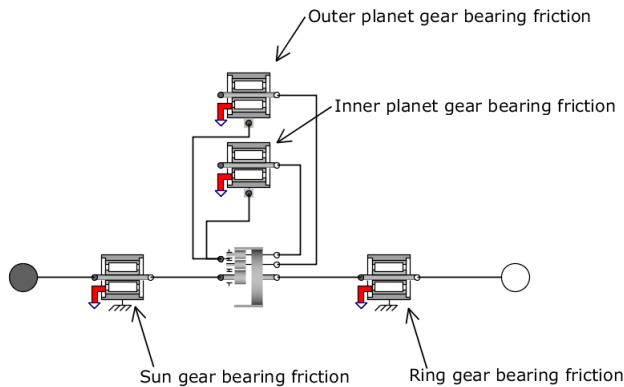


Figure 6: Adding bearing friction to gear sets

3 Advantage of the Symbolic Technology

Symbolic techniques turn out to be a critical ingredient, both to enable efficient modeling of these components as well as to generate optimized code, yielding the required HiL execution speed. MapleSim’s symbolic capabilities are enabled by an underlying Maple computation engine [6], providing extremely efficient symbolic operations that are necessary for handling the thousands of system equations typically found in the transmission models.

A common characteristic of Modelica environments is that system models are built by assembling components using “physical” connections, carrying quantities like torque and rotational angle bidirectionally between the two components. The decision on causality of the model is deferred to simulation time, just before the numeric integration process is started. This is possible because the entire set of equations for the whole system is generated symbolically, as a first step. At this point we typically have a set of differential algebraic equations. As shown in Figure 7, several steps are necessary before these equations can be solved numerically, yielding simulation results and/or HIL code. These steps are discussed next.

3.1 Equation Simplification

The initial set of equations generated from the system model is typically large and contains many redundancies. Symbolic techniques are used to simplify this set of equations as much as possible. The simplifications are exact and do not result in any loss of fidelity in the model. Trivial equations of the form $a = b$ are removed. Linear equations are pre-solved analytically. Reducing the number of equations by a factor of ten is not uncommon. This simplification

step is key to the scalability of the remaining pre-processing steps.

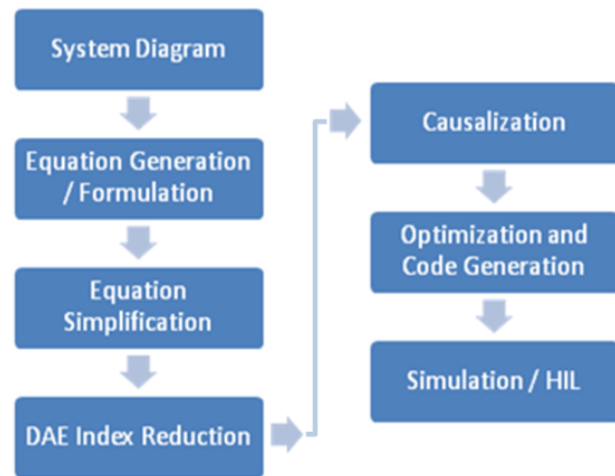


Figure 7: Steps towards numerical simulation

3.2 Index Reduction

The generated system consists of differential algebraic equations (DAEs). Such equations cannot be readily solved with standard numerical techniques because of the presence of algebraic constraints. The “index” of a DAE is loosely defined as the number of times the equations need to be differentiated in order to remove these constraints. The goal here is to reduce the system of equations to “index 1”, allowing numeric integration. During integration, the constraints are monitored for “drift”, ensuring an accurate solution, reflecting the behaviour of both the differential equations as well as algebraic constraints. Again, symbolic techniques turn out to be essential, allowing differentiation of equations and efficient index reduction.

3.3 Causalization

At this point, we have a simplified system of (index 1) differential equations. In order to numerically solve this system, we will need to repeatedly evaluate the system for a particular point. To enable this, we will need to turn our (acausal) system of equations into a (causal) sequence of numeric operations. In short, this process involves imposing an order of evaluation onto our set of equations. Doing this efficiently involves tools from graph theory, readily available in the symbolic computing tool chest.

3.4 Optimized Code Generation

Executing speed is critical to HiL applications and symbolic techniques again turn out to be key to gen-

erating highly efficient code. It is, of course, possible to generate code directly from the causal system of equations described above. However by looking at those equations globally, we are able to perform symbolic optimizations prior to generating code, which makes the difference between achieving the required HiL cycle times or not. These optimizations involve detecting common computation sequences that can be factored out, which go way beyond the (local) optimization capabilities of available compilers.

3.5 Two Examples

A Simple Driveline Model:

Consider the driveline mode shown in Figure 8. The model represents a vehicle powertrain from engine to dynamometer. The model includes a torque converter between the flywheel and the transmission. The transmission is a 4-speed Ravigneaux gearbox. Using throttle and brake controllers, the speed is changed following a ramp-up/coast down profile.

Using MapleSim's API commands from Maple, the simulation time is measured. A fixed time-step solver (Euler) is used here with a time step of 0.001 sec. Total simulation time is 150 seconds. The simulation was done on a 64-bit Windows 7 machine with Intel(R) Core(TM) Duo 2.40 GHz CPU. Figure 9 shows Maple's commands for this example. These command extract and simplify the model equations and convert them to optimized c-code. The simulation results are obtained from a Maple procedure which includes the compiled c-code.

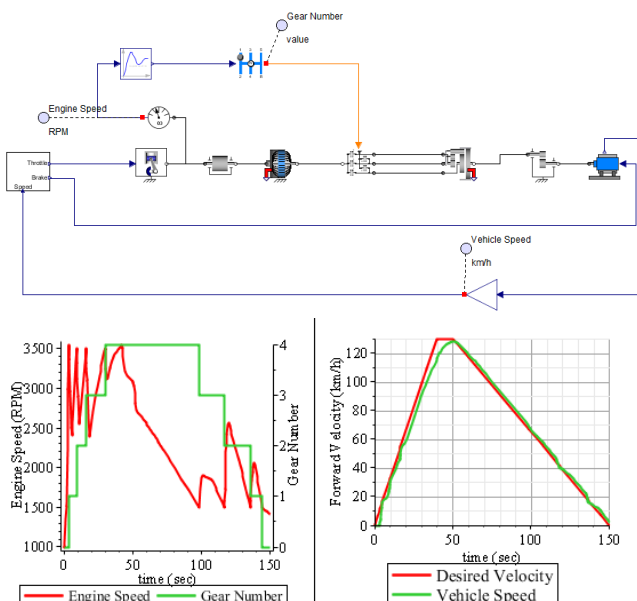


Figure 8: An example of a complete powertrain.

The simulation was done over 15 times faster than real-time (i.e. ~10 second of integration time for a 150-second simulation). In 20 consecutive runs the average simulation time was 9.68 with standard deviation of 0.30.

```
> A := MapleSim:-LinkModel( ) :
> cProc := A:-GetCompiledProc(params = [Main_kl = 1]) :
> st := time( ) :
  results := cProc(duration = 150, dt = 0.001) :
  time( ) - st
```

9.609

Figure 9: Running MapleSim simulation using API commands in Maple

A Vehicle Model with Mean-value Engine Model:

The system in Figure 10, is the second example chosen for the real-time demonstration. This model is considerably more complex than the previous example and includes a detailed mean-value engine model [7] and a 4-speed transmission model. The MapleSim model uses the New York City Cycle [8] and runs for 600 seconds. Simulation timing was done under similar solver settings as the previous example. The same computer was also used. On average the simulation was done about 12 times faster than real-time (i.e. ~50 second of integration time for a 600-second simulation). Based on 15 consecutive runs the average simulation time was 50.2 seconds with standard deviation of 0.54.

4 HiL Simulation of the Automatic Transmissions

At AISIN AW, HiL simulation is extensively used to accelerate the development of automatic transmissions. The plant models for HiL simulations require sufficiently high fidelity to accurately represent the aspect of the system dynamics important to the designers. At the same time, these models have to have low calculation cost in order to enable real-time execution.

As shown in Figure 11, the real-time platform used in the HiL simulations reported here is the ADX system [9] from A&D Technology, Inc.

The plant model is deployed in Simulink [10] and can be separated into two parts as depicted in Figure 12. The first part is the plant model which is constructed of the s-function generated from MapleSim models including clutches, brakes, and various gear sets. This part also includes Simulink blocks for other parts of plant model. The second part is the automatic testing module.

It is critical that the calculation time associated with the first part (plant model) is kept as low as

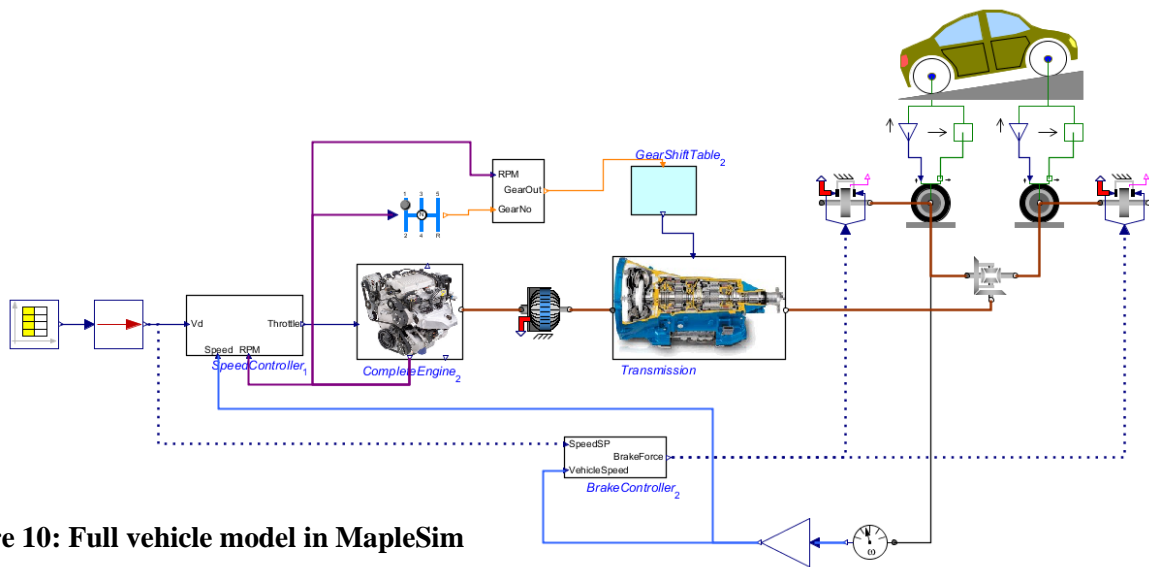


Figure 10: Full vehicle model in MapleSim

possible to accommodate for the high execution times of the increasingly more complex automatic testing routines implemented in the testing module.

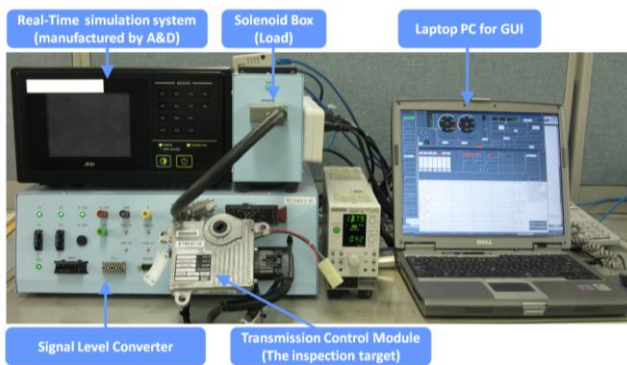


Figure 11: HiL simulation system

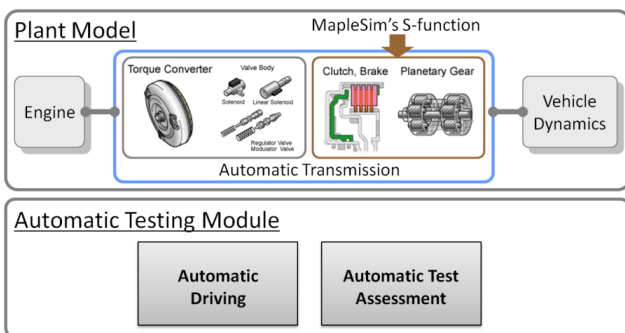


Figure 12: Model for Real-time system

A sample gear train is shown in Figure 13 which includes a *planetary gear*, a *Ravigneaux gear*, and a *basic gear* connected together using three *clutches*, two *brakes* (modeled using clutch components), and a *one-way clutch*. This gear train is connected to an *ideal gear* which represents the differential gear ra-

tio. The tire load is modeled using additional inertia, clutch, and brake components. The *tire component* and the *longitudinal vehicle dynamics component* (refer to Figure 1) are not used here since that level of fidelity is not necessary for the intended HiL simulations.

Figure 14 shows the HiL simulation results with s-function generated from MapleSim. Compared with another software previously used, it was shown that for the above model, the implementation of the s-function generated from MapleSim in the HiL simulations with a sampling time of 1ms, reduced the overall CPU time by 250 μ s (or 25% of a time step). This reduction is due to fact that the MapleSim's s-function runs twice as fast as the previously implemented block.

5 Conclusions

In this paper some of the features of the Driveline Component Library – an add-on Modelica library for MapleSim modeling, simulation, and analysis environment – were introduced. The Driveline Component Library provides a comprehensive set of components that enable transmission manufacturers – as well as other automotive developers – to conveniently create plant models for control and simulation. The underlying symbolic computation engine of MapleSim (i.e. Maple) expands the inherent advantages of similar Modelica-based physical modeling tools to new heights. Benefiting from the power of symbolic computing, MapleSim can generate extremely fast code that is of vital importance when simulating large complex systems in real-time.

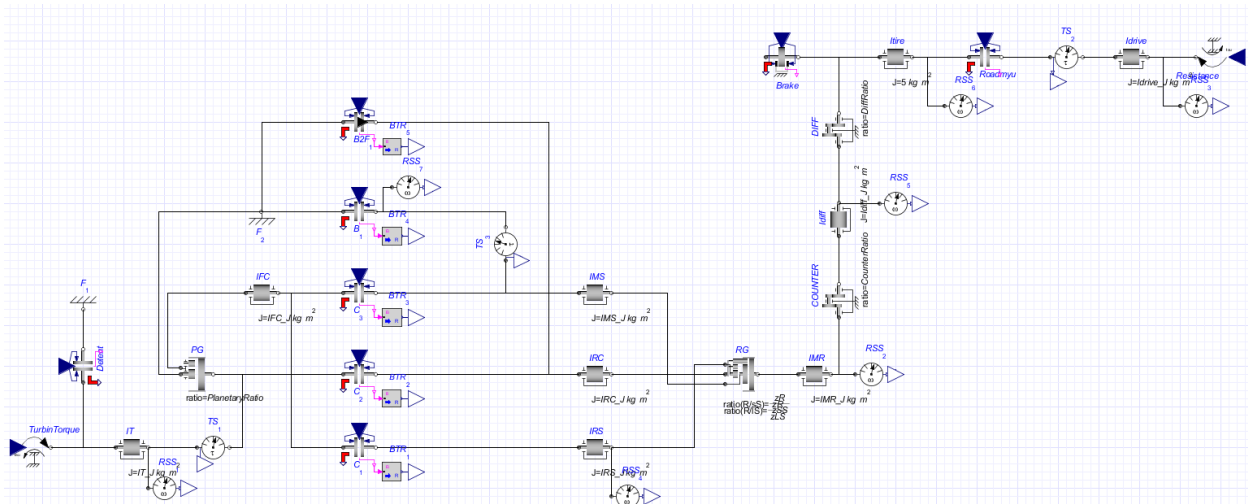


Figure 13: Gear train model created in MapleSim

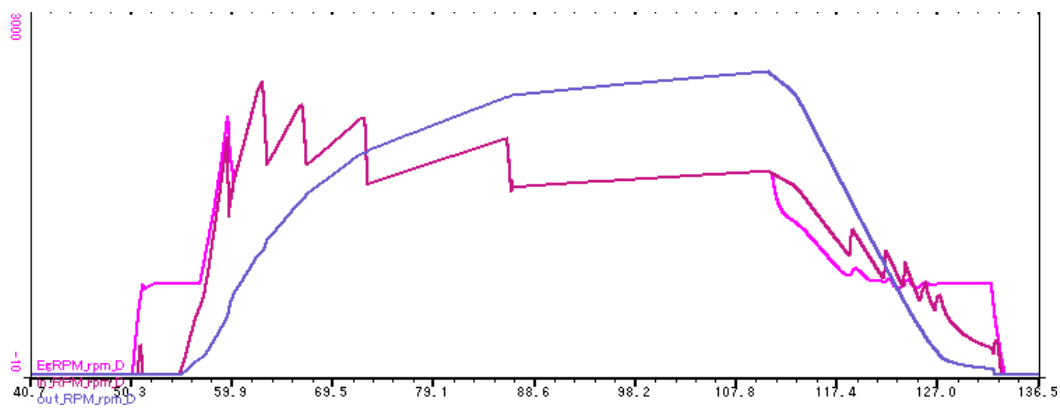


Figure 14: A sample of HiL simulation results

The paper also included a brief description of the activities at AISIN AW on the development of new automatic transmissions and their use of MapleSim and the Driveline Component Library in HiL simulations. The optimized c-code generated by MapleSim from transmission plant models enabled AISIN AW to perform more detailed HiL simulations. In a sample case study, it was shown that the s-function generated by MapleSim ran twice as fast as the s-function generated by a similar tool.

References

- [1] *MapleSim User's Guide*, 2011, ISBN 978-1-926902-09-8.
- [2] <https://www.modelica.org/> (accessed 2/4/2012).
- [3] D. Hrovat and W.E. Tobler. "Bond graph modeling and computer simulation of automotive torque converters," *Journal of the Franklin Institute*. Volume 319, Issues 1-2, January-February 1985, pp 93-114.
- [4] Pelchen C., Schweiger C., and Otter M., "Modeling and Simulating the Efficiency of Gearboxes and of Planetary Gearboxes," *2nd International Modelica Conference*, Proceedings, pp. 257-266.
- [5] Joško Deur, Vladimir Ivanovic', Matthew Hancock, and Francis Assadian. "Modeling and Analysis of Active Differential Dynamics," *Journal of Dynamic Systems, Measurement, and Control*, 2010. Volume 132 / 061501, pp 1-14.
- [6] Bernadin L., Chin P., DeMarco P., Geddes K. O., Hare D. E. G., Heal K. M., Labahn G., May J. P., McCarron, Monagan M. B., Ohashi D., and Vorkoetter S. M., *Maple Programming Guide*, 2011, ISBN 1-926902-08-1.
- [7] -, "Mean-Value Internal Combustion Engine Model", *Maplesoft, White Paper*, <http://www.maplesoft.com/contact/webforms/whitepapers/enginemodel.aspx>, (accessed: 2/4/2012).

- [8] -, *Dynamometer Driver's Aid*,
<http://www.epa.gov/nvfe/testing/dynamometer.htm>, (accessed: 2/4/2012).
- [9] <http://www.aanddtech.com/ADX.html> (accessed: 2/4/2012).
- [10] <http://www.mathworks.com/products/simulink/> (accessed: 2/4/2012).

ADGenKinetics: An Algorithmically Differentiated Library for Biochemical Networks Modeling via Simplified Kinetics Formats

Atiyah Elsheikh
Austrian Institute of Technology, Vienna, Austria

Abstract

This work demonstrates the compact but powerful freely available Modelica library *ADGenKinetics* for descriptive modeling of biochemical reaction networks using simplified kinetics formats. While existing powerful works and guidelines for modeling biochemical reaction networks based on classical mechanistic kinetics already exist, in this work a first attempt of utilizing the power of Modelica constructs for providing a compact implementation of simplified kinetic formats with generalized structured formulas is presented. This gives the opportunity of realizing biochemical reaction networks using few number of reaction components, in contrast to libraries based on classical mechanistic kinetics which require hundreds of reaction components. Moreover, *ADGenKinetics* is the first algorithmically differentiated Modelica library that is enhanced with differentiated components by which parameter sensitivities are additionally computed with minimal efforts from the user perspective.

Keywords: enzyme kinetics, biochemical reaction networks, systems biology, algorithmic differentiation

1 Introduction

Modelica as a universal modeling language with a lot of capabilities for supporting hierarchical modeling, multidisciplinary modeling, object-oriented reusable components and different modeling flavours with a large degree of freedom and creativity is continuously attracting the attention of many scientific fields. However, in the field of Systems Biology aiming at studying cellular process with the aid of mathematical models, there are still few published non-standardized attempts for modeling biochemical reaction networks for describing the metabolism within cellular activities, one of the core modeling activities demanded by many applications of the field of Systems Biology.

This work demonstrates a comprehensive overview of the compact Modelica library *ADGenKinetics* for a specific set of reaction kinetics. These subsets of enzyme kinetics are referred to as simplified kinetics formats and are represented by generalized structured kinetics formulas suitable for biochemical reactions with arbitrary number of substrates, products, inhibitors and activators. The employment of generalized kinetics have two advantages from two perspectives:

1. From the modeling perspective: Utilization of generalized kinetics formulas provides the opportunity of implementing a compact library with so few numbers of components that the user neither needs to choose an enzyme kinetic component from a long list of components nor needs to self implement newer enzyme kinetics for newer cases of non considered reactions
2. From the implementation perspective: By efficient employment of powerful Modelica language constructs, the implementation of highly specialized practical library for modeling biochemical network applications gets simplified.

The proposed library is distinguished by the following criteria:

- It is suitably adequate to get linked with specialized graphical editors for modeling biochemical reaction networks and for other applications of automatic model generation
- It is the first algorithmically differentiated library by which algorithmic differentiation (AD) techniques [6, 11] are directly applied at the library level [5]. The resulting additional subpackage contains extended components in which parameter sensitivities, i.e. derivatives of model variables w.r.t. model parameters, are represented.

- it is open-source and provided under the Modelica License 2.

The rest of the work is structured as follows: section 2 presents a quick introduction to biochemical networks modeling. Section 3 gives rather a quick overview of various classical and simplified kinetic laws but comprehensive enough for appreciating this contribution. Section 4 demonstrates the proposed library, its advantages and limitations along an example in section 5. Finally, outlook is given in section 6.

2 Background and Terminology

2.1 Modeling biochemical reaction networks

Biochemical reaction network models are used for describing the dynamics of molecular species and their interaction within the cellular metabolism [9]. Usually such models are based on the continuum¹ and homogeneity assumptions². The law of mass conservation³ is used for describing the rate of change in the mass of intermediate metabolites (i.e. biochemical substances) in a biochemical reaction network. The resulting models typically have the following structure:

$$\dot{c} = N \cdot v(c, \alpha), \quad c(0) = c_0 \quad (1)$$

where $c \in R^m$ stands for vectors of the metabolite concentrations, $v = v(c, p) \in R^n$ is a vector of reaction rates described by enzyme kinetics, α is kinetic parameters vector describing enzyme characteristics and $N \in R^{m \times n}$ is the reactions stoichiometry describing the number of participating molecules in any single reaction [15].

Figure 1 demonstrates a typical biochemical network of enzymatic reactions termed as the Spirallus which represents an abstraction of Tri-Carboxylic Acid (TCA) cycle [14]. The set of freely distributed metabolites A, B, C, D, E, F are viewed as nodes, while the reactions are viewed as intermediate edges among the metabolites [2]. With the presence of substrates being taken up through the initial reaction v_{upt} , intermediate reactions become active and the two products E_{ex}, F_{ex} get produced as long as enough substrate molecules are taken up. Some of the reactions are *irreversible* such as v_3 (i.e. the flow of materials is conceptually only in the forward direction) while others

¹All chemical species involved have such a high copy number to be described by a continuous concentration value

²Diffusion processes are so fast that concentrations can be considered to be spatially homogeneous

³the mass within a closed system remains constant over time

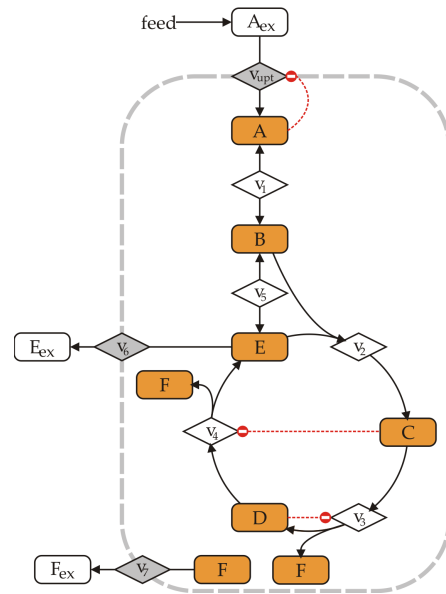


Figure 1: Spirallus: An Abstraction of the TCA cycle

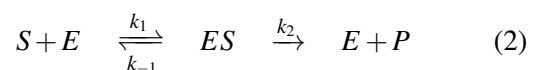
are *reversible* such as v_1 . The reactions v_{upt}, v_3, v_4 are *inhibited* by the molecules of the metabolites A, D, C acting as *inhibitors* respectively. Analogously, specific molecules of metabolites may act as *activators* by accelerating certain reactions. Inhibitors and activators are referred to as *effectors* or *modifiers*. A mathematical model for describing the process dynamics is given as follows:

$$\begin{aligned} \dot{A} &= v_{upt} - v_1 & \dot{B} &= v_1 - v_2 - v_5 \\ \dot{C} &= v_2 - v_3 & \dot{D} &= v_3 - v_4 \\ \dot{E} &= v_4 + v_5 - v_2 - v_6 & \dot{F} &= v_3 + v_4 - v_7 \end{aligned}$$

The state variables A, B, \dots, F_{ex} corresponds to the concentration of metabolites. The algebraic variables v_j describe the reaction rates via algebraic functions referred to as *enzyme kinetics*, the topic of the following section.

3 Enzyme kinetics

Vital cellular processes at the metabolism level are performed according to the present set of enzymatic reactions networks. The base elements of such networks are the involved enzymatic reactions. For instance, within an uni-uni *enzymatic reaction* in the form:



the molecules of the specific enzyme E binds with the molecules of the substrate S according to a rate constant k_1 . Similarly, k_{-1} is a rate constant describing the decomposition rate of the complex ES into E and S . The resulting enzyme-substrate complex ES molecules are vastly transformed to the product P . The reaction rate v of such transformation, i.e. product formation, is modeled by *enzyme kinetics*. Such kinetics typically correspond to nonlinear functions of the following form:

$$v(t) = e \cdot f(c(t), \alpha) \quad (3)$$

where e is the amount of the associated enzyme, α a set of parameters corresponding to enzyme characteristics and $c(t)$ the concentration of the involved substrates, products and effectors [1]. In case of a reversible reaction, f can be usually expressed in terms of forward and backward reaction rate as $v = v^{fwd} - v^{bwd}$. In this case, the overall direction of the reaction is then the sign of v . Many *enzyme kinetics* approaches for describing the function f exist some of which are demonstrated in the following subsections.

3.1 Mechanistic kinetics

In order to emphasize the importance of simplified kinetics formats, the widely used classical mechanistic kinetics are introduced as a motivation. Mechanistic kinetics describe the reaction rates of biochemical enzymatic reactions by involving the underlying enzyme binding mechanisms within the mathematical model. For instance, the simple reaction (2) is modelled by *Michaelis-Menten* kinetic. Its analytical derivation based on the quasi-steady state assumption (i.e. $k_{-1}, k_1 \gg k_2$) leads to the following formula:

$$v = \frac{k_2[E]_0[S]}{\frac{k_{-1}+k_2}{k_1} + [S]} = \frac{V_{max}[S]}{K_m + [S]} \quad (4)$$

The parameter K_m corresponds to the substrate concentration that yields the half-maximal reaction rate $V_{max}/2$. These two parameters represent enzymatic characteristics demonstrating how quickly the enzyme becomes saturated and what its maximum activity is.

Reactions with effectors

For enzymatic activities influenced by effectors various types and binding mechanisms exist, cf. figure 2 for various inhibition mechanisms. Mechanistic kinetics distinguish such types of inhibitions mechanisms through their mathematical formulation according to whether

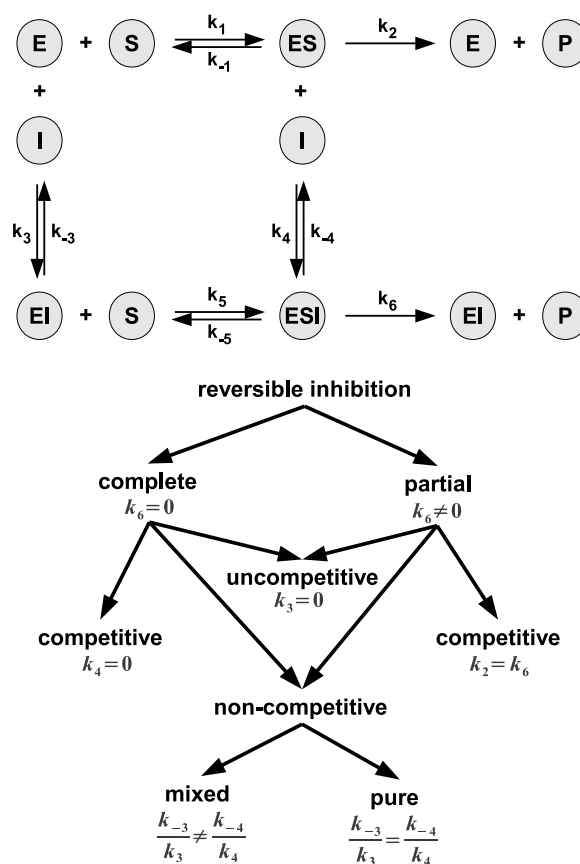


Figure 2: A summary of different types of inhibition mechanisms

- the inhibitor binds to the complex ES
- the inhibitor binds to S
- the reversibility of the inhibition

For instance, the analytical derivation of a mechanistic kinetic for an irreversible reaction inhibited by I according to complete competitive inhibition leads to:

$$v = \frac{V_{max}[S]}{K_m(1 + [I]/K_I) + [S]} \quad (5)$$

Where K_I is a parameter that expresses the ratio of EI formation to EI decomposition. Within competitive inhibition, the inhibitor I competes with the substrate S for binding with the enzyme E . In this case, the release of P is blocked by I , cf. figure 2. Similar discussion regarding activation mechanisms leads to the fact that a wide range of mechanistic kinetics formulas exists distinguishing all these various mechanisms.

Multi-substrate reactions

For cellular reactions with more than one substrate and one product, very likely to arise in the cellular

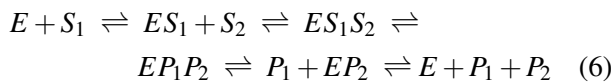
metabolism, mechanistic kinetics are more sophisticated. Their analytical derivation additionally considers the sequence in which substrates bind and products are released. For example, within a bi-bi reaction, (two substrates S_1, S_2 and two products P_1, P_2) the underlying enzymatic mechanisms are differentiated according to whether binding to enzyme is done

- in random order, (i.e. E binds with both of S_1 and S_2 in any order)
- in a sequential order, (i.e. S_2 binds only with the complex ES_1)
- in an alternate binding of substrates and release of products (ping-pong mechanisms)

as well as

- which intermediate complexes are formed (only ES_1, ES_2 or also ES_1S_2)
- Interactions among reactants (e.g. inhibition through product formation)

For example, the kinetic of an ordered bi-bi reaction (i.e. binding in a specific order)



is described with the equation:

$$v = \frac{V_{max}[S_1][S_2]}{K_{iS_1}K_{mS_2} + K_{mS_2}[S_1] + [S_1][S_2]} \quad (7)$$

In summary, each combination of assumptions regarding the underlying enzymatic reaction leads to a unique kinetic formula. This results in enormous number of possible equation patterns corresponding to combinatorially high number of different assumptions. Such equations do not necessarily follow a general equation pattern if they are expressed in terms of mechanistic parameters rather than elementary rate constants k_i . This causes some difficulties by modeling since hundreds of components need to be separately implemented for expressing different enzyme binding mechanisms.

3.2 Generalized kinetics formats

As already shown, mechanistic kinetics characterize detailed description of the underlying enzymatic mechanism. These kinetics pose however some problems when used for describing enzymatic reactions

within cellular environment. Under such crowded conditions, a lot of effectors may influence the enzyme activity. When considering all typical interactions, the corresponding derived kinetic becomes very complex and parameter dependencies are enhanced when estimating the parameters with experimentally generated data leading to serious problems in the process of model identification [16].

This argument motivates the use of generalized kinetics which rely on more simplified assumptions two of which are introduced. The first type is the so-called *convenience kinetics* which assumes a reversible rapid equilibrium with random binding mechanism [10]. In this way, the corresponding kinetic of any reaction with arbitrary number of substrates S_i , products P_j , inhibitors I_b and activators A_a becomes:

$$v = \prod_a \frac{K_{A_a} + [A_a]}{K_{A_a}} \cdot \prod_b \frac{K_{I_b}}{K_{I_b} + [I_b]} \cdot \frac{V_{max}^{fwd} \prod_i \frac{[S_i]}{K_{mS_i}} - V_{max}^{bwd} \prod_j \frac{[P_j]}{K_{mP_j}}}{\prod_i \left(1 + \frac{[S_i]}{K_{mS_i}}\right) + \prod_j \left(1 + \frac{[P_j]}{K_{mP_j}}\right) - 1} \quad (8)$$

Another kinetic format is the *linlog* kinetic [8] given by:

$$v = v^0 + \sum_i \alpha_i \cdot \ln\left(\frac{S_i}{S_i^0}\right) + \sum_j \beta_j \cdot \ln\left(\frac{P_j}{P_j^0}\right) + \sum_a \gamma_a \cdot \ln\left(\frac{A_a}{A_a^0}\right) + \sum_b \delta_b \cdot \ln\left(\frac{I_b}{I_b^0}\right) \quad (9)$$

In contrary to mechanistic parameters, which provide descriptive physical insights into enzymatic mechanisms, linlog parameters are based on scaled sensitivities describing the influence of characteristic changes of enzymes on a referenced reaction rate at a reference steady-state v^0 .

One of the main advantages of the presented kinetics in the context of this work is that they are expressed in terms of generalized structured formulas very adequate for compact implementation and automatic generation of highly complex models. However, one of the limitation of such kinetics is that they may not describe the enzymatic behavior accurately in some boundary cases as shown in [7].

4 Overview of the library

4.1 The Biochem Library

For modeling biochemical networks with Modelica, considerable efforts have been realized by the *Biochem* library [12]. It provides the essential guidelines and design principles for achieving this goal, eg. basic implementable interfaces and basic types. According to the available publications, *Biochem* provides about 99 abstract reaction types under the restriction that a reaction can get connected to at most three substrates, three products and one effector. Out of these abstract types, many mechanistic reaction kinetics can be derived. Within the library *Metabolic*, a published implementation of *Biochem*, at least 180 kinetics are implemented and classified according to the number of substrates and products within many sub-packages. If all combination of reaction assumptions are considered, still many hundred of reaction kinetics need to be inserted. If more than one effector is considered, a realistic scenario for biochemical reaction networks in cellular environment, the number of required components corresponding to various kinetics would be so high.

4.2 The ADGenKinetics Library

In this work, a compact implementation of simplified kinetics is demonstrated following the main guidelines provided by *Biochem* w.r.t. the library structure, physical units, naming conventions and some of the implementation. The main differences appear whenever the mathematical structures of the simplified kinetics are utilized for implementing interfaces for the underlying generalized formulas. These interfaces are specialized according to the number of reactants, products, specific effectors, reversibility etc. By exploiting powerful Modelica constructs, realization of simplified kinetics require very few number of components out of which realistic biochemical networks are easily constructed, modeled and simulated. On the other hand classical mechanistic kinetics within implementation of *Biochem* requires a large number of components. Users are likely to insert additional kinetics whenever new or slightly modified biochemical reaction networks need to be modeled.

Figure 3 summarizes the presented library. The following packages are available:

- *Interfaces*: connectors, classification interfaces

and icons

- *NodeElements*: components for nodes
- *Reactions*: components for reactions
- *Derivatives*: extended components for computing parameter sensitivities
- *Examples*: biochemical network models

Further two subpackages within *NodeElements* and *Reactions* exist corresponding to two ways of declaration of connectors within components:

1. *dynamic*: parametrized number of connections
2. *static*: fixed number of connections

The main differences of both ways and their advantages and disadvantages are emphasized in this section along with the given examples. Common interfaces and abstract classes are located above these packages.

4.3 Connectors

The fundamental laws on which biochemical reaction network models rely i.e. the continuum and homogeneity assumptions and the law of conservation (cf. section 2.1) translates into the terminology of Modelica as follows:

Listing 1: Implementation of chemical ports

```
connector ChemicalPort
  "reaction connector from a node to
  a reaction"
  Units.Concentration c "Concentration";
  flow Units.VolumetricReactionRate r
    "reaction rate";
end ChemicalPort;
```

That is, the concentration of a substance is the potential variable while the flow rate of materials represents the flow variables when connecting nodes and reactions together. The mathematical models of biochemical reaction networks do not require a node to distinguish between a connection from an ingoing reaction and a connection to an outgoing reaction. The sign of the reaction rate is explicitly determining whether the considered node is a substrate or a product of the connected reaction.

This situation is different with connections to nodes from a reaction side. The kinetic formula distinguishes between a substrate node and a product node, cf. equation (8). Consequently additional connectors, *ChemicalPort_S*, *ChemicalPort_P* extending the connector

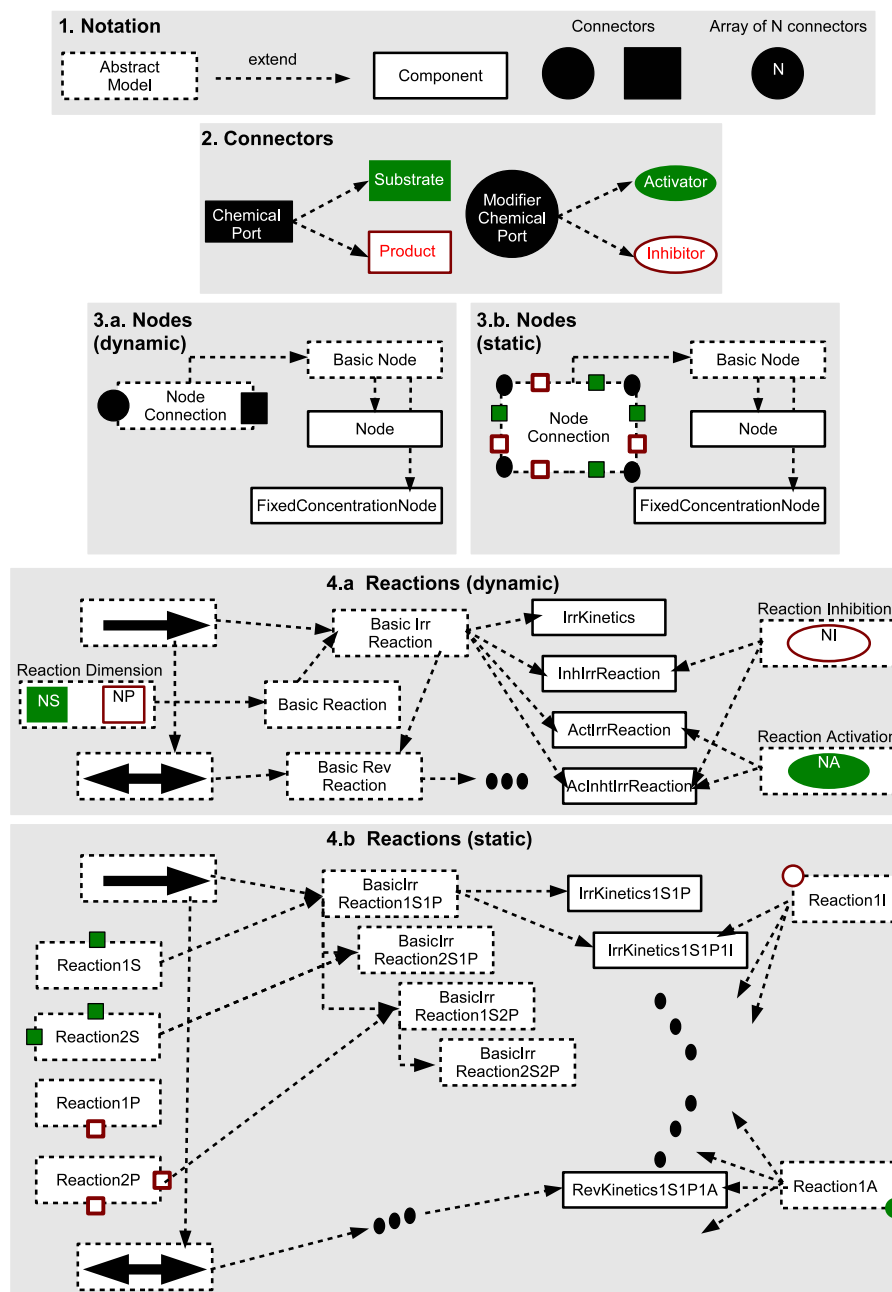


Figure 3: An overview of the library with static/dynamic number of connections

ChemicalPort with distinguished icons for differentiating between connections to substrates and connections to products are considered. Additionally, the specialized connector *ModifierChemicalPort* between effectors and reactions is provided. This connector includes only the concentration of the respective node. Similarly two icons are provided to distinguish activators from inhibitors.

4.4 Nodes

In *Biochem*, implementation of nodes is realized at three levels of abstraction:

1. *NodeConnections*: A class providing the basic interfaces and icons, about 8 connectors as a reactant and 4 connectors as a modifier
2. *BasicNode*: An abstract class realizing basic implementation of nodes and extending *NodeConnections*
3. *Node*: An implementation of *BasicNode* describ-

ing the concentration dynamics

The *static* subpackage is directly taken from *Biochem*.

In the subpackage *dynamic*, only one connector for reactants and one connector for modifiers [15] is given as follows:

Listing 2: Implementation of node connections

```
partial model NodeConnections
  "Metabolite connections to reactions"
  Interfaces.ChemicalPort rc
    "connection to any reaction ";
  Interfaces.ModifierChemicalPort mc
    "connection as a modifier";
end NodeConnections;
```

The abstract class *BasicNode* in *dynamic* looks as follows:

Listing 3: Implementation of an abstract node class

```
partial model BasicNode
  "Basic declarations of any Metabolite"
  extends Interfaces.dynamic.
    NodeConnections;
  parameter Units.Concentration c_0=0;
  Units.Concentration c(start=c_0);
  Units.VolumetricReactionRate r_net;
equation
  r_net = rc.r;
  rc.c = c;
  mc.c = c;
end BasicNode;
```

Direct implementation of *BasicNode* is realized in the models *FixedConcentrationNode* and *Node*:

Listing 4: Implementation of a node

```
model Node "Metabolite with dynamic rate"
  extends NodeElements.dynamic.BasicNode;
equation
  der(c) = r_net;
end Node;
```

Further types of nodes exist in *Biochem*.

4.5 Reactions

Each generalized kinetic format is realized within a subpackage. Currently, the subpackage *convenience* is implemented. The realization of other simplified kinetic formats like linlog kinetics is analogously straightforward.

4.5.1 *dynamic*

In this subpackage, *convenience* kinetics are implemented by extending several abstract classes which specifies a reaction according to:

1. its dimension: how many substrates and products are involved as well as the stoichiometry of the reactants
2. its reversibility
3. whether the reaction is effected by other modifiers, how many and their types

The implementation of these basic classes are shown as follows:

Listing 5: The dimension of a reaction

```
class ReactionDimension
  "Dimension and structure of a reaction"
  parameter Integer NS = 1
    "Number of substrates";
  parameter Units.StoichiometricCoef
    n_S[NS]=ones(NS)
    "Stoichiometry of all substrates";
  parameter Integer NP = 1
    "Number of products";
  parameter Units.StoichiometricCoef
    n_P[NP]=ones(NP)
    "Stoichiometry of all products";
end ReactionDimension;
```

Using the previous class, an abstract type for reactions slightly modified version from the one provided in *Biochemis* given as follows:

Listing 6: The dimension of a reaction

```
partial model BasicReaction
  "basic declaration of a reaction"
  extends Interfaces.dynamic.Dimension
    .ReactionDimension;
  Units.VolumetricReactionRate v
    "reaction rate";
  Interfaces.ChemicalPort_S rc_S[NS]
    "connection to substrates";
  Interfaces.ChemicalPort_P rc_P[NP]
    "connection to product";
equation
  rc_S[:].r = n_S[:] * v;
  rc_P[:].r = -n_P[:] * v;
end BasicReaction;
```

Specification of the reaction reversibility is done via the related classes *OneWayReaction* and *TwoWayReaction*. These classes provide the basic declaration of related kinetic parameters and are directly taken from *Biochem*. Moreover, two additional abstract classes *BasicIrrReaction* and *BasicRevReaction* are introduced in the proposed library for emphasizing type abstractions among implemented kinetics:

Listing 7: Basic reversible reaction

```
partial model BasicRevReaction
  "basic implementation of
  a reversible reaction"
```

```

extends Reactions.convenience.dynamic.
  BasicIrrReaction;
extends Interfaces.Reversible.TwoWay;
Real P1 "Product terms nominator";
Real P2 "Product terms denominator";
parameter Units.AffinityConst KmP [NS]
  = ones(NS) "Affinity constants of
  the product node";
equation
  P1 = Vbwdmax * product({rc_P[i].c/KmP[i]
    for i in 1:NP});
  P2 = product({rc_P[i].c/KmP[i] + 1
    for i in 1:NP});
end BasicRevReaction;

```

The corresponding classes for specifying the effectors are given by the classes *ReactionInhibition* and *ReactionActivation*:

Listing 8: The inhibitors of a reaction

```

partial model ReactionInhibition
  "Inhibition influencing a reaction"
  parameter Integer NI = 1
  "# Metabolites inhibiting the reaction";
  Interfaces.ModifierChemicalPort_I
    mc_I [NI];
  parameter Units.AffinityConst KI [NI]
    = ones(NI) "Affinity constant";
  Real I "Inhibition term";
equation
  I = product({KI[i] / (KI[i] + mc_I[i].c)
    for i in 1:NI});
end ReactionInhibition;

```

Using these classes, all reaction types of convenience kinetics are realized only with 8 classes. For instance, the implementation of convenience kinetics for reversible inhibited reactions with arbitrary number of reactant substrates, products and inhibitors is given as follows:

Listing 9: Kinetic for reversible inhibited reaction

```

class InhRevKinetic
  "S1+S2+... <==I1,I2,..==> P1,P2,..."
  extends Reactions.convenience.dynamic.
    BasicRevReaction;
  extends Reactions.convenience.dynamic.
    ReactionInhibition;
equation
  v = I * (S1 - P1) / (S2 + P2 - 1);
end InhRevKinetic;

```

Realistic biochemical reaction networks can be realized using only these 8 classes.

4.5.2 static

This subpackage is more or less a straightforward implementation of the *Biochem* guidelines except in

some details. It provides many components for describing enzyme kinetics with fixed number of substrates, products and modifiers via a static number of connectors. The implementation is done along many levels via the following abstract classes:

1. The classes *Reaction1S*, *Reaction2S*, ... , *Reaction1P*, *Reaction2P*, ... , *Reaction1I*, *Reaction1A* etc. provide the basic icons for reactions with specific number of connectors to substrates, products, etc.
2. The abstract classes *BasicIrrReaction1S1P*, *BasicIrrReaction2S1P*,...etc. provide basic implementation for kinetic terms of irreversible and reversible reactions with specific number of reactants. Similarly, the classes *BasicReaction1I*, *BasicReaction1I* provide basic implementation for kinetics terms of modified reactions.
3. The actual kinetics are realized within *IrrKinetic1S1P*, *IrrKinetic1S2P*, ... , *IrrKinetic1S1P1I*,... etc. by extending and specializing the abstract classes.

Using this way, many components need to be provided. For instance, by realizing reactions with two substrates and two products together with one modifier at maximum, there are 2 (substrates) \times 2 (products) \times 2 (reversibility) \times 3 (effectors) = 24 components that need to be provided. By three substrates and three products with two modifiers at maximum, about $9 \times 2 \times 6 = 108$ components need to be provided.

4.6 The Derivatives subpackage

The *Derivatives* subpackage contains an extended copy of the whole library with identical structure of subpackages, interfaces and components. Each component has additional declaration and equations for computing parameter sensitivities. The equations are computed using algorithmic differentiation techniques. In this work, new novel AD techniques especially optimized for equation-based languages are employed. For any model using the library typically corresponding to a DAE system of the form:

$$F(\dot{x}, x, p, t) = 0 \quad , \quad x(t_0) = x_0(p) \quad (10)$$

where $x(t) \in R^n$ and $p \in R^m$ represent state variables and model parameters, respectively, importing the types within the *Derivatives* subpackage lets the underlying model of eq. (10) get extended with the

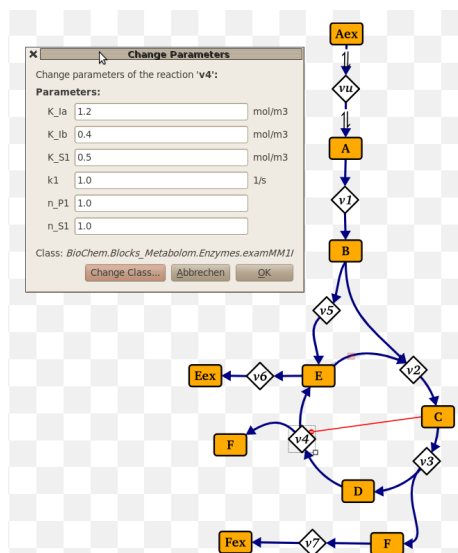


Figure 4: Omix: a highly-specialized graphical editor for biochemical networks

corresponding sensitivity subsystems:

$$[F_x \dot{s}_i + F_x s_i + F_p] J_p = 0 \quad , \quad s_i(t_0) = \frac{\partial x_0(p)}{\partial p_i} \quad (11)$$

$$\text{where } s_i = \frac{\partial x}{\partial p_i} \quad \text{for } i = 1, 2, \dots, m$$

and $J_p \in R^{m \times r}$ is the input Jacobian specifying the set of active parameters $q \in R^r$ w.r.t. which derivatives are sought. The same model simulates the underlying biochemical reaction network together with the derivatives of all variables with respect to the specified input parameters. A usage example is available in the *Examples* subpackage and is summarized in the following section.

5 Examples

The implementation of the biochemical network in figure 1 is demonstrated once with dynamic number of connections and again with static number of connections. With dynamic number of connections, the implementation is assembled as follows:

Listing 10: Implementation of the Spirallus network with parametrized number of connections

```

model Spirallusdyn
  "An abstraction of the TCA cycle"
  import ADGenKinetics.
    NodeElements.dynamic.*;
  import ADGenKinetics.
    Reactions.convenience.dynamic.*;

  Node Aex(c_0=1);

```

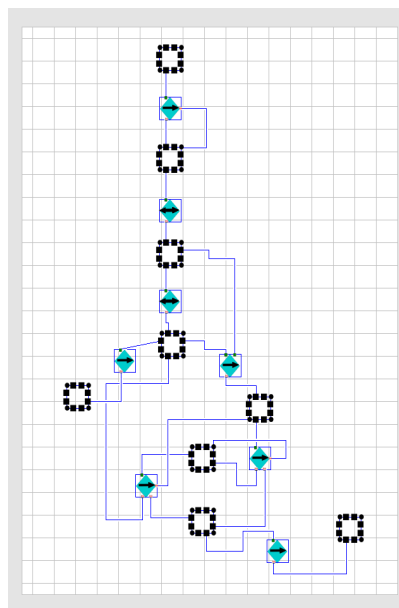


Figure 5: Implementation with the Dymola graphical editor

```

InhIrrKinetic vupt(NS=1,NP=1,NI=1,
  Vfwdmax=1.0,
  KmS={0.1},KI={3.0});
ModifierNode A;

RevKinetic v1(NS=1,NP=1,
  Vfwdmax=3.0,Vbwdmax=1.0,
  KmS={0.1},KMP={3.0});

Node B;
...
equation

// vupt
connect(Aex.rc,vupt.rc_S[1]);
connect(vupt.rc_P[1],A.rc);
connect(vupt.mc_I[1],A.mc);

// v1
connect(A.rc,v1.rc_S[1]);
connect(v1.rc_P[1],B.rc);
...
end Spirallusdyn;

```

The main disadvantage of this approach is that the implementation is provided only at textual level. Typical modeling environments of Modelica don't currently provide graphical support for parametrized dynamic number of connections yet. Nevertheless, this approach is ideally relevant for automatic model generation possibly using specialized graphical editors for biochemical networks. For instance, figure 4 shows a snap shot of Omix [2] a general-purpose editor for constructing, editing and visualizing biochemical networks in a semi-automatic manner.

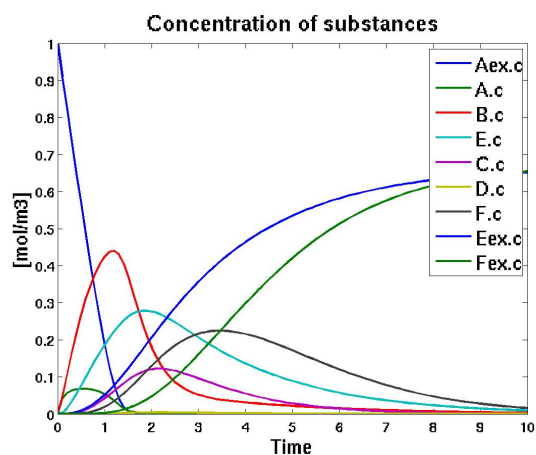


Figure 6: Concentration of the substances

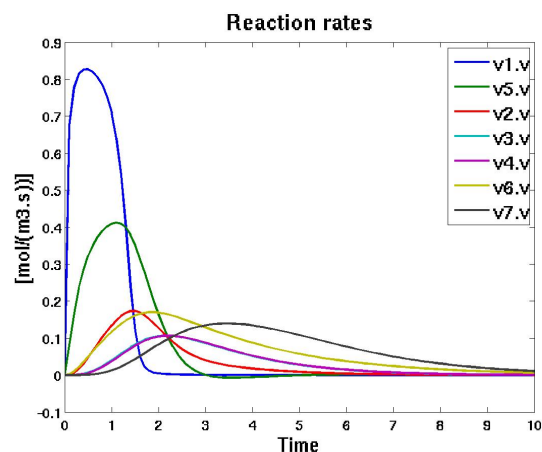


Figure 7: Reaction rates of reactions

Omix is enhanced with a plugin for parsing and generating Modelica models [13]. The tool employs Open Modelica Compiler (OMC) for parsing a Modelica library for biochemical network modeling and identifying existing types of kinetics and nodes. Then OMC is again used for automatically generating the corresponding models that can be then simulated using typical Modelica simulation environments as described in [3] in a very similar manner to the tool provided in [4]. The presented library would be ideal for such a tool or any other SBML-based graphical editor using very similar concepts.

With the static components of fixed number of connections, biochemical network models can be directly assembled with common Modelica simulation environments. For instance, figure 5 provides the implementation of the network model using Dymola. Figures 6 and 7 demonstrate the simulation results of the concentration of chemical substances and the reaction rates of reactions of the Spirallus network, respectively. Using the subpackage *Derivatives*, parameter sensitivities can be computed in a straight forward way. For the Spirallus example, this can be done by slightly modifying the declaration part of the code from listing 10 to the following:

Listing 11: Implementation of the dynamics of the Spirallus network together with the parameter sensitivities

```
import ADGenKinetics.Derivatives.
  NodeElements.dynamic.*;
import ADGenKinetics.Derivatives.
  Reactions.convenience.dynamic.*;
import ADGenKinetics.Derivatives.
  Functions.*;
```

```
inner parameter Integer NG = 24
  "Number of gradients";

Node Aex(c_0=1);
InhRevKinetic vupt(NS=1,NP=1,
  Vfwdmax=1.0,g_Vfwdmax=unitVector(1,NG),
  KmS={0.1},g_KmS={unitVector(2,NG)},
  KI={3.0},g_KI={unitVector(3,NG)});
...
IrrKinetic v7(NS=1,NP=1,
  Vfwdmax=2.0,g_Vfwdmax=unitVector(23,NG),
  KmS={3.0},g_KmS={unitVector(24,NG)});
Node Fex;

equation
  // equations remain as before
  ...
```

In the last model, the standard types for nodes and reactions are replaced by the extended types within the subpackage *Derivatives*. An additional unique parameter NG is declared, specifying the number of active parameters w.r.t. which derivatives are sought. Finally, the input gradient of any parameter p is initialized with the help of the function $unitVector(i,NG)$ which returns a unit vector of length NG with the i th component equal to one. In this way, for any variable v , $g_v[i]$ corresponds to $\partial v / \partial p$. For parameters with non-initialized gradients, they simply become inactive. Figure 8 shows the parameter sensitivities of the reaction v_7 w.r.t. all kinetic parameters.

6 Outlook

In this work, a Modelic library for implementing generalized kinetics formats based on justifiable simplification assumptions is provided. With the help of Modelica language constructs, the opportunity of real-

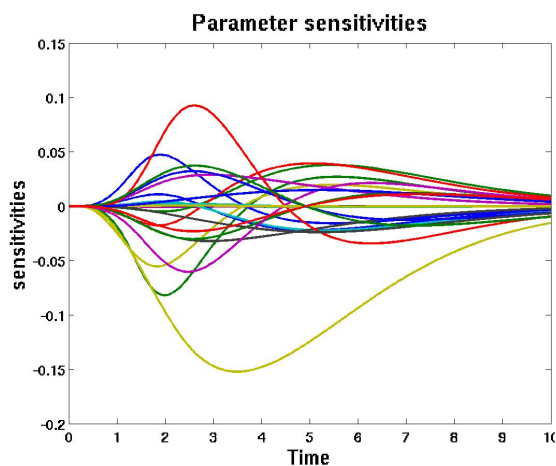


Figure 8: Parameter sensitivities of v_7

izing real-life applications with few number of components is given. Consequently, the library is especially adequate for tools requiring automatic model generation. Moreover, this library follows the main guidelines of Biochem making it possible to get integrated with other existing implementation. The presented library is the first algorithmically differentiated Modelica library. With minimal additional user efforts, base models additionally simulate parameter sensitivities together with the network dynamics. The underlying novel equation-based AD techniques which have been especially designed for *ADGenKinetics* have also the potentials to be employed by other Modelica libraries.

Acknowledgement

I'd like to acknowledge Dr. Stephan Noack, institute of Bio- and Geosciences, Biotechnology (IBG-1), research centre Jülich, for valuable discussions about enzyme kinetics. His own advanced modeling library was very inspiring for me for understanding the principles of modeling biochemical reaction networks.

References

- [1] H. Bisswanger. *Enzyme Kinetics, Principles and Methods*. WILEY-VCH Verlag, Weinheim, Germany, 2002.
- [2] P. Droste, S. Noack, K. Noh, and W. Wiechert. Customizable visualization of multi-omics data in the context of biochemical networks. In *VIZ 2009: The 2nd International Conference on Information Visualisation*, Barcelona, Spain, 2009.
- [3] A. Elsheikh. *Modelica-based computational tools for sensitivity analysis via automatic differentiation*. PhD thesis, submitted to Institute of Computer Science, RWTH Aachen University, Aachen, Germany, 2011.
- [4] A. Elsheikh, S. Noack, and W. Wiechert. Sensitivity analysis of Modelica applications via automatic differentiation. In *Modelica'2008: The 6th International Modelica Conference*, Bielefeld, Germany, 2008.
- [5] A. Elsheikh and W. Wiechert. Automatic sensitivity analysis of DAE-systems generated from equation-based modeling languages. In C. H. Bischof, H. M. Bücker, P. D. Hovland, U. Naumann, and J. Utke, editors, *Advances in Automatic Differentiation*, pages 235–246. Springer, 2008.
- [6] A. Griewank. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. Number 19 in Frontiers in Appl. Math. SIAM, Philadelphia, PA, 2000.
- [7] F. Hadlich, S. Noack, and W. Wiechert. Translating biochemical network models between different kinetic formats. *Metabolic Engineering*, 11(2):87 – 100, 2009.
- [8] J. J. Heijnen. Approximative kinetic formats used in metabolic network modeling. *Biotechnology and Bioengineering*, 91(5):534–545, 2005.
- [9] E. Klipp, R. Herwig, A. Kowald, C. Wierling, and H. Lehrach. *Systems Biology in Practice: Concepts, Implementation and Application*. Wiley-VCH, 2005.
- [10] W. Liebermeister and E. Klipp. Bringing metabolic networks to life: convenience rate law and thermodynamic constraints. *Theoretical Biology and Medical Modelling*, 2006.
- [11] U. Naumann. *The art of Differentiating Computer Programs, an Introduction to Algorithmic Differentiation*. SIAM, 2012.
- [12] E. L. Nilsson and P. Fritzson. A Metabolic Specialization of a General Purpose Modelica Library for Biological and Biochemical Systems. In *Proceeding of the 4th International Modelica Conference*, Hamburg, Germany, 2005.

- [13] J. Tillack, P. Droste, N. Hackbarth, W. Wiechert, and K. Nöh. Visually-assisted modeling of kinetic metabolic networks - from Omix to Modelica and back. In *MATHMOD 2012: The 7th Vienna International Conference on Mathematical Modelling*, Vienna, Austria, 2012.
- [14] S. A. Wahl. *Methoden zur integrierten Analyse metabolischer Netzwerke unter stationären und instationären Bedingungen*. PhD thesis, Research Centre Jülich, Germany, 2007.
- [15] W. Wiechert, S. Noack, and A. Elsheikh. Modeling languages for biochemical network simulation: Reaction vs equation based approaches. *Advances in Biochemical Engineering / Biotechnology*, 2010.
- [16] W. Wiechert and R. Takors. Validation of metabolic models: Concepts, tools, and problems. In H. V. Westerhoff and B. Kholodenko, editors, *Metabolic Engineering in the Post Genomic Era (Horizon Bioscience)*. Horizon Scientific Press, 2004.

Variable Structure Modeling for Vehicle Refrigeration Applications

Imke Krüger Alexandra Mehlhase Gerhard Schmitz

Hamburg University of Technology, Department of Technical Thermodynamics
Denickestr. 17, 21075 Hamburg

TU Berlin, Department of Software Engineering and Theoretical Computer Science
Ernst-Reuter-Platz 7, 10587 Berlin

Abstract

A variable-structure approach for Modelica models is presented in this paper. Variable structure models enable the user to change the simulation model during runtime. This is not supported by common simulation environments and thus a Matlab script is used to control the run of the simulation. The script switches between the different models and sets the initial values to ensure smooth transients of the variables. The method is applied to a simplified model of a thermal management system for Lithium ion batteries in a hybrid vehicle. In this model some components do not need to be calculated through the complete simulation time and are removed from the model through the variable-structure approach. With this approach the simulation time can be reduced while the simulation accuracy is not affected negatively.

Keywords: vapour compression cycle; simulation speed; thermal management, variable-structure model

1 Introduction

How can the variable-structure method help to speed up simulations? In the case of battery thermal management, the branch to the battery cooling can be opened or closed with a valve such that the battery is only cooled when needed. So the general structure of the refrigeration cycle changes from a branched cycle to a single evaporator cycle. In simulation environments supporting Modelica it is not possible to change the set and causality of an equation system. In Modelica it is assumed that a model always has one set of equations and that the variables themselves do not change. For the refrigeration cycle it would be highly useful to be able to change the equation system because the equations for the unneeded branch could be turned off. This means that no unnecessary calculations have to be

done and the simulation time could be reduced. To explain this approach the thermal management of HEV batteries will first be explained. Then the general approach for variable-structure models that was used in this paper is introduced. The presentation of a simple model and its preparation for the application of the variable-structure method is followed by the results for simulations with and without the variable-structure method.

2 Thermal Management of HEV batteries

The batteries of hybrid electric vehicles heat up due to inner heat generation. Thermal management is though essential to ensure safety and prevent ageing. The only reliable heat sink for the cells is the automotive refrigeration cycle. The cells can be cooled by evaporation of the refrigerant, therefore a cold plate is put in parallel to the ordinary evaporator (see fig.1).

Cooling of the cells is only necessary when their upper temperature limit is reached. Only then the valve to the battery cooling branch opens, e.g. there is no refrigerant flow as long as the cells are cool enough. The necessary cooling power depends on the drive cycle and the surrounding temperatures.

System simulation plays an important role in the design of vehicle air conditioning. It enables the user to test various system architectures as well as providing values that cannot be measured in real life test rigs. As the development becomes faster and additional tasks like battery cooling emerge, accelerating the simulations becomes necessary. Additional components and more complex system designs raise the dimensions of the resulting nonlinear equation systems. During the evaluation process of a cooling system, a lot of simulations for various climatic conditions and heat loads

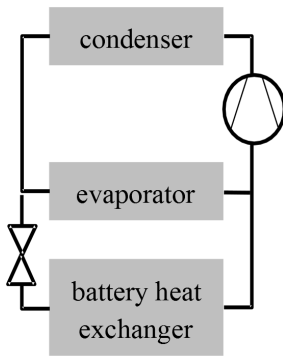


Figure 1: System architecture of A/C Refrigerant Cycle with Battery Cooling

are necessary to evaluate the additional energy consumption of the refrigeration cycle.

In the current Modelica models with static structure, the refrigerant mass flow in the battery branch cannot be set to exactly zero. The resulting very small refrigerant mass flows and pressure losses in the control volumes slow down the simulation. In addition, the mass flow might change its sign, causing further deceleration of the simulation. The equations for the closed branch have to be solved during the whole simulation although they are not needed most of the simulation time. The CPU time needed for simulations becomes too large and the number of possible simulation runs is limited by the available time. Very simple (and less exact) models have to be used, making the results less reliable.

Calculation time could be radically reduced if the obsolete equations could be switched off when the battery branch is closed. The time span during which the valve is closed can make up large parts of a driving cycle (see [1]) so there is a large potential to reduce the time for a simulation run. Currently there is no possibility to deactivate equations in Dymola/Modelica during runtime.

3 Variable-structure modeling with Modelica/Dymola

A variable-structure model consists of different modes whereas each mode itself is a model and has a set of equations and variables. The model can switch from one mode to another triggered through a switch-

ing condition. When a switching condition occurs the mode switch takes place and the end values of the simulation results are used to initialize the next mode. The modeler has to define which end values to use to initialize the next mode.

As explained above such a change of an equation system is needed to model the thermal management of batteries more efficiently. But neither Dymola nor other simulation environments e.g. OpenModelica and SimulationX support the change of a set of equations of a Modelica model during a simulation run. Therefore, a scripting approach with Matlab is used as introduced in [2]. This approach allows a user to model their models in a chosen simulation environment or language and use Matlab to switch from one mode (and therefore to another set of equations) to the next.

The general idea is to create a new modeling layer where the structural change is described and which handles the actual change. The simulation models are implemented in a simulation environment chosen by the user. It is important that the simulation environment can be controlled through Matlab so a model can be compiled and a simulation run can be started using Matlab.

Figure 2 illustrates the sequence of operations in a Matlab script that handles the change of a set of equations of a model. In this example the variable-structure model has two modes, which means we have two models whereas each model has its own set of equations. For this example we use Dymola as a simulation environment, but other environments could be integrated as well.

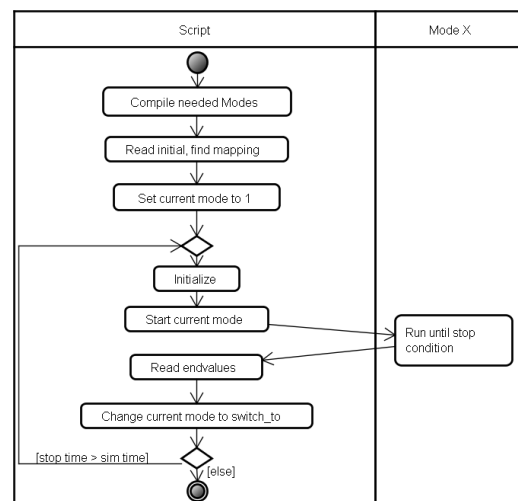


Figure 2: Course of events of a Matlab script to change a set of equations of a Dymola model

As a first step all needed models are compiled,

which means an executable model 'dymosim.exe' with an initialization file 'dsin.txt' is created which can be started through Matlab. Afterwards the simulation parameters (start time, stop time, solver, etc.) are set.

Then a mapping of variables takes place. In this mapping process the initialization files of all modes are loaded. These files contain all variables and parameters with their startvalues. An equivalent file is created by Dymola at the end of each simulation of a model (called 'dsfinal.txt'), containing the endvalues of variables. When loading such a file we get an array with all variable names and an array with all initial (or end) values of a simulation.

For a mode transition between two modes it is necessary to map user defined variables from the end-array to the initial-array of the next mode. Therefore, a mapping matrix is created for each transition. This matrix holds the indices of the values to be read from the end-array in the first column and the indices of the values to be overwritten in the initial-array in the second column.

This mapping matrix is created at the beginning because it saves simulation time when a transition is needed more than once, for instance switching from mode 1 -> 2 -> 1 -> 2 would mean that the mapping matrix from mode 1 to mode 2 can be used twice. After this preparation phase is done the simulation of the first mode is started. The script uses the dymosim.exe which is created when compiling a Dymola model to simulate the model.

When a defined stop condition is reached, which is implemented in the model itself, a terminate command will stop the simulation. As soon as the simulation terminates, the end values of the simulation are read from the 'dsfinal.txt' file. The earlier created mapping matrix for this transition is then used to map the simulation results to the initial data in the dsin.txt file for the next mode.

The script then starts the dymosim.exe of the second model. This simulation runs until the stop time or another terminate condition is reached. Then the script again processes the simulation data and either the stop time of the simulation is reached which stops the simulation completely or the script changes back to the first mode via a transition and the mapping matrix in this transition.

With this simple approach Dymola can be used to simulate variable-structure models even though Dymola on its own does not support these kind of models. This means that existing Modelica models can be reused for variable-structure models and that they do

not have to be remodeled in other tools or languages as SOL [3], MOSILAB [4] or Hydra [5] which do support variable-structure modeling to a certain extent. The problem with these approaches is, that SOL is an experimental language and does not support index reduction and solvers in the extent that Dymola does. MOSILAB does not support index reduction at all and is not freely available for it is still under development. Hydra is based on functional programming languages and is therefore not as easy to learn for modelers. All the existing approaches would mean a remodeling of the existing air condition models.

4 Evaluation

4.1 Model

All models in this use case are based on the AirConditioning Library by Modelon [6], based on the AC lib [7].

A simple test case was created to evaluate the variable-structure approach. It reduces the complex model of a refrigeration cycle with thermal management of the battery to the main components that are affected when the battery cooling branch is closed. The model consists of an evaporator with a discretized pipe in parallel, the branch to the pipe can be closed with a valve. The valve has a variable Kv-value that can be set by an input source. The original model with all initial equations activated (Figure 3) serves also as the initial mode for the variable-structure model. For the

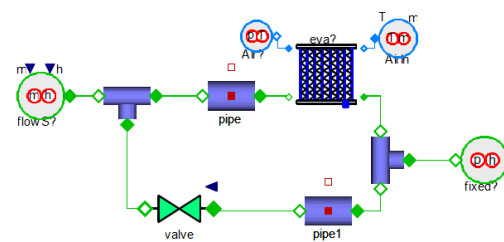


Figure 3: model for mode 1 with two branches

second mode (Figure 4), only the evaporator branch remains. Two new component models were created: splitResistance and junctionResistance. They represent the pressure loss in the corresponding components from the first mode. Using the same names for the components eases the mapping of the parameters and start values when a switch occurs. This means at the beginning of the Matlab script where the mapping takes place, all components and their variables which

have the same name (e.g. evaporator) are mapped in the mapping matrix of the transition.

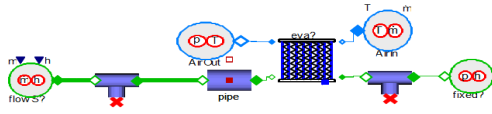


Figure 4: Modelica model for mode 2 with only one branch

The simulations are carried out as follows: The valve is closed with a ramp function, beginning at 30s simulation time. After 10s, the valve is closed. At 50s (10s after complete closure) the variable-structure model switches to mode 2. At 150s the variable-structure model switches back to mode 1, the valve opens again at 160s. Figure 5 shows the sequence for the kv-Value of the valve.

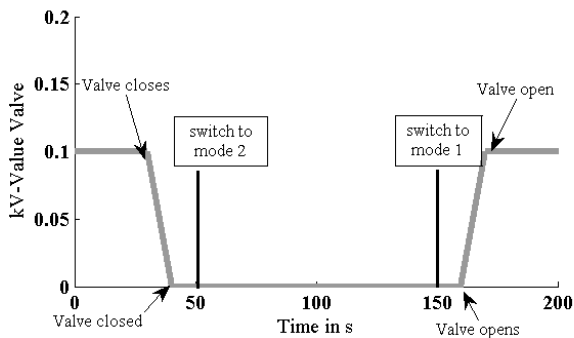


Figure 5: Simulation sequence for the kv-value of the valve

4.2 Preparation of the modes

As described above the mode 1 model has more components than the mode 2 model but each component from mode 2 has been in the first model, too. Therefore, it is known that the end values of the first model can be used to initialize all components of the second model. To make the identification of the components which exist in both modes easier, they were called exactly the same. Therefore, the mapping function called at the beginning of the Matlab script can create a mapping matrix which maps all variables existing in both modes to each other. To create the mapping matrix the script takes the lists of the initial names of both modes and searches through this list to match the names. This is necessary because the order of the variables might not be the same, even (sub-)variables of a component such as (evaporator.p[1], evaporator.[T]) might be in different order. The first mode has about 7

400 variables and the second mode about 6 900 variables which makes this mapping process time consuming. A better mapping algorithm is planned for future work.

If the variables and components are not called the same, the modeler can define which variables and components belong together. For instance the modeler can define that all variables from a component 'a' from mode one have to be matched to all variables from a component 'b' in model two. In this case all variables inside these components are matched and the mapping is saved in the mapping matrix.

To be able to initialize the second model through the Matlab script the model needs to be prepared for a script initialization. Many components in the AirConditioning Library are per default set to initialization through parameters and initial equations. This leads to the problem that an initialization from outside is not per default possible. For instance look at the following model:

```
model init
  Real T1(start = 100);
  Real T2;
  initial equation
    T2 = T1-10;
    ...
end init;
```

In this example T1 has a start value of 100. But T2 can only be initialized through T1 and is 10 smaller than T1. With such a model we are not able to initialize the T's separately because we cannot ignore the initial equation even though this might be necessary if this model is the second mode and different values are needed. In the AirConditioning library such cases can be handled by setting the `initType` of the components to 'noInit'. If a component cannot be initialized externally and does not have such an `initType` provided the user has to change the model to use it in a variable-structure model. Often these initial equations are deep down in the model hierarchy, e.g. temperature of the evaporator wall and therefore it is quite difficult to locate all needed changes. This does not mean that initial equations are not allowed for variable-structure models, it just means that a modeler has to know what his model is doing and if the initial equations hinder an initialization from the outside.

An easy way to test if the model can be initialized though an extern file is to simulate a model for a period of time and use the end values from the 'dsfinal.txt' as initial file and restart the simulation of the same model.

If the simulation results are smooth around the mode change it is usually save to assume that the initialization worked.

A problem with the initialization through Matlab is that when using the given Matlab methods to handle Dymola the initialization does not always work. For instance, it is possible to use a method `dymosim.m` which gets as parameters the name of the Dymola model and the initial values (and some other data). But this method does not seem to write the `dsin.txt` (more precisely the `dsin.mat` as it is called from Matlab) correctly. This means the initialization does not work correctly and the simulation results are wrong.

First it was assumed that the initial equations in the AirCondition model were the problem but it was discovered that the given Matlab function seems to be the problem. Therefore a new initialization method in Matlab was written. This method uses the mapping matrix of the transition and creates a new initialization matrix for the new mode, which only holds the end data of the old mode and user defined values. All other data is not included in this initialization matrix. This new initialization matrix is then saved in the models `dsin` file. When switching back from the second mode to the first not all necessary data is known to initialize the mode. Therefore, the modeler has to specify the additional values separately in the Matlab script.

4.3 Results

The results of the variable structure model are conform to a large extent with the results of the static structure model, which calculates the unnecessary branch through the whole simulation time. The mass flow in and out of the split can be seen in figure 6. The valve starts to close at 30s, the mass flow changes according to the changing pressure drops in the branches. When the valve is completely closed, the mass flow into the battery branch is almost zero but shows still little variations around zero for the static structure model.

The variation of the mass flow results in variations of the pressure drop in the junction. Figure 7 and 8 show the inlet and outlet pressure of the evaporator. The little variations are thus propagated to all the components of the model, to the complex ones (in this example the evaporator), too.

The needed CPU time for the simulation runs is plotted in figure 9. The CPU time is given through Dymola and is the time from calling the `dymosim.exe` until exiting the simulation. Until the first switch of the variable-structure model, the CPU times rise with the same speed for both simulations. The second mode

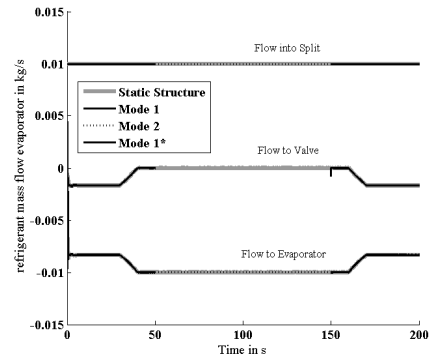


Figure 6: Refrigeration mass flow in and out of the split for static and variable structure model

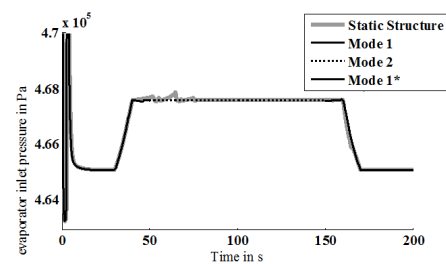


Figure 7: Refrigeration pressure at evaporator inlet

of the variable-structure model is calculated rather fast and does only need a short simulation time. While the variable-structure model does only need a short simulation time during this phase where only one branch of the model is simulated the static structure model needs a lot of simulation time. The opening of the valve at 160s lets the CPU time of the static model rise almost vertically. Whereas the variable-structure models CPU time does not rise that high. This already shows that while only simulating a short period of time (200 seconds), it is already possible to save a lot of simulation time through the variable-structure approach. In this example both modes were simulated for the same period of time.

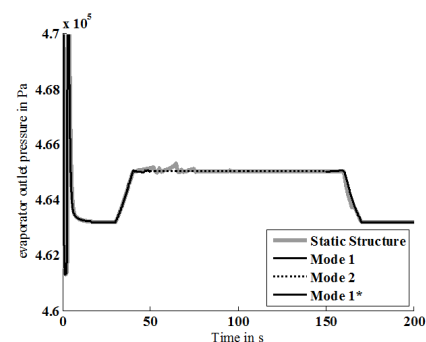


Figure 8: Refrigeration pressure at evaporator outlet

If, as was mentioned in the introduction, the two branched model is only needed for short periods during a drive cycle a lot of simulation time can be saved. But it also has to be considered that using a variable-structure model also means that a switching procedure is necessary and that each mode needs to be compiled. Compiling the two necessary modes of the variable-structure models takes about 22 seconds whereas the compilation of the static structure only takes about 13 seconds. Creating the mapping matrix for each transition at the beginning of the script take about 10 seconds – the search algorithm is not optimized yet and the time could be significantly reduced with a better algorithm.

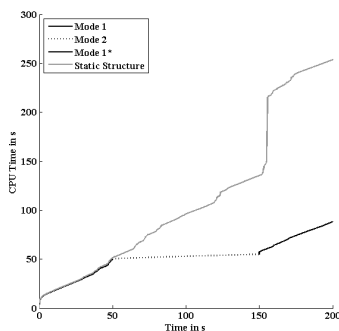


Figure 9: CPU time needed for simulation

The switching from one mode to the next with loading the end values and setting the initial values takes about 0.5 seconds per switch. This means that when looking at the overall simulation time of the presented example the variable-structure model needs about 121 seconds whereas the static structure models takes about 270 seconds. This means that even with the necessary overhead the simulation with the variable-structure model is still faster.

4.4 Restrictions

As already mentioned throughout the paper the variable-structure approach has some restrictions and the modeler has to regard certain points. At first the initialization needs to be mentioned, the modeler needs to define how the end values of one mode are used to initialize the next mode. If the old mode does not provide enough data for the initialization the modeler either has to provide the missing data (by concrete values or calculations) or the variable-structure approach might not be feasible. Furthermore, the models used as modes need to be initialized from the outside, so the modeler might need to adapt the models to

fulfill this requirement.

As it also is with conventional modeling it is with variable-structure modeling, too, that one should not use it just because one can. For instance in conventional modeling a model with few equations might suffice even though one could model it more accurately which might result in solver problems or time problems. So it is with variable-structure modeling. In some cases the approach might be usable but not feasible, because the switch does not lead to a significant positive effect. For instance the simulation time is not reduced and the accuracy is the same. Another possibility is that the switch is done at the wrong time, e.g. switching to the second mode of the refrigeration application when the valve is not closed yet. This will lead to a model with inconsistent results. This means that, as in conventional modeling, the modeler needs to know his models and what he wants to do to use the variable-structure approach feasible and sensible.

5 Summary

System simulation for refrigeration cycle models in vehicle refrigeration applications is time consuming. The variable structure method presented in this paper can help to reduce the needed CPU time and the overall simulation time for such a model.

With the help of a Matlab script, the user can switch between several representations of the same model.

It takes some time to prepare and test the models. If the given advices are already considered during modeling the method can be easily used to speed up simulations.

The method can be applied to other applications with variable-structure with more than two modes, too. A Python framework which guides the user through the steps to describe a variable-structure model is currently worked on. This will enable the user to describe the models more easily and to use a free software (Python) instead of Matlab. Furthermore, more simulation environments will be integrated so the user is not limited to Dymola.

It is planned to investigate the advantages of variable-structure models more thoroughly and with more complicated models. With these researches it will be possible to find out when variable-structure models can be used sensibly and when it is more useful to use a static structure model.

References

- [1] Krüger, I. Energy Consumption Of Battery Cooling In Hybrid Electric Vehicles. In: Proceedings of 14th International Refrigeration and Air Conditioning Conference, Purdue, USA, 16-19 July 2012 (to be published).
- [2] Mehlhase A. Varying the level of detail during simulation. In: Proceedings of ASIM 2011, Symposium Simulationstechnik, Winterthur, Swiss, 7-9 September 2011.
- [3] Zimmer D. Equation-Based Modeling of Variable-Structure Systems. Ph.D. thesis, Swiss Federal Institute of Technology, 2010.
- [4] Nytsch-Geusen, C., Ernst, T., Nordwig, A., and et al. (2005). Mosilab: Development of a modelica based generic simulation tool supporting model structural dynamics. In: G. Schmitz (ed.), Proceedings of the 4th International Modelica Conference, Hamburg, March 7-8, 2005.
- [5] Nilsson, H., Giorgidze, G. (2010). Exploiting structural dynamism in Functional Hybrid Modelling for simulation of ideal diodes. In: Proceedings of the 7th EUROSIM Congress on Modelling and Simulation. Czech Technical University Publishing House, Prague, Czech Republic, 2010.
- [6] Tummescheid, H., Eborn, J., Prölß, K., AirConditioning - a Modelica library for dynamic simulation of AC systems, In: G. Schmitz (ed.), Proceedings of the 4th International Modelica Conference, Hamburg, March 7-8, 2005. vol. 1, p. 185-192.
- [7] Pfafferoth, T., Schmitz, G., Modeling and transient simulation of CO₂-refrigeration systems with Modelica, International Journal of Refrigeration, 2004, Vol. 24, no. 1, p.42-52.

Thermal Simulation of Power-Controlled Micro-CHP Systems for Residential Buildings

Sebastian Stinner Dirk Müller
Institute for Energy Efficient Buildings and Indoor Climate
E.ON Energy Research Center, RWTH Aachen University
Mathieustr.10, 52074 Aachen, Germany
sstinner@eonerc.rwth-aachen.de

Abstract

Combined heat and power (CHP) plants are a well-known technology for industrial and district heating appliances. As those plants are often used to optimally satisfy thermal demands they often run heat-controlled. The power generation profiles of those plants are badly predictable. Those badly predictable power generation profiles are fluctuating and central power plants have to work in the times when the heat-controlled plants do not run. Due to these circumstances it should be analysed to what extent a power-controlled operation can be applied. For this purpose a dynamic simulation of the whole system is necessary. This paper presents the possibilities of a dynamic simulation of a one-family-house with a power-controlled micro-CHP unit and a thermal storage.

Keywords: CHP; electrical grid; grid compatibility

1 Introduction

The energy supply of Germany will change significantly within the next few years. An increasing part of the power supply will be based on fluctuating sources like wind power or photovoltaics. On the other hand, about 35 per cent of the final energy in Germany is used for space heating and domestic hot water in buildings [3]. Thus, reasonable concepts for the building sector have to be found.

In periods where there is not enough power supply from the regenerative sources, flexible and energy efficient alternatives have to be considered. One of these alternatives could be combined heat and power (CHP) plants. The waste heat of the power generation process is used at the same time for example for space heating and domestic hot water. In many cases these plants can work more efficient than other options to generate heat. To combine the advantages of the CHP-plant

as a fast reacting power generator and as an efficient heat supply system, those plants should be operated in a power-controlled way. This includes that a thermal storage is needed to buffer the discrepancy between the run times of the CHP plants and the thermal demand in the building.

Because of the increasing dynamics of those systems, dynamic simulations have to be considered to evaluate which role a system with power-controlled CHP-plants can play in the future. For these simulations, libraries are used that were developed at the Institute for Energy Efficient Buildings and Indoor Climate at RWTH Aachen University. They offer the possibility to simulate the performance of the CHP plant and the storage system, the dynamic thermal characteristics of the building and the user behaviour [2]. With this approach, an integrated evaluation of whole micro-CHP systems is possible.

2 Definition of the problem

Due to new regulations (e.g. [1]) and an increased awareness of energy topics in public, the integration of regenerative sources in the German power supply increases. Especially wind power and photovoltaics are used as regenerative power sources. The disadvantage of these technologies is their non-controllability. They are completely dependent to weather conditions (wind speed and directions, solar radiation) and it is clear that there is a discrepancy between generation and demand.

At some times, there will be more regenerative energy generated than needed. At other times, the demand is higher than the generation from renewable energies. This would especially happen, if there are many consecutive days without wind and maybe nearly no direct solar radiation because of clouds and

other effects. The arising gap between generation and demand should be closed with technologies that are as efficient as possible. (Micro-)CHP units, for instance, can reach higher efficiencies than pure power generating units (figure 1), and could be a suitable alternative to conventional power plants. Another aspect is

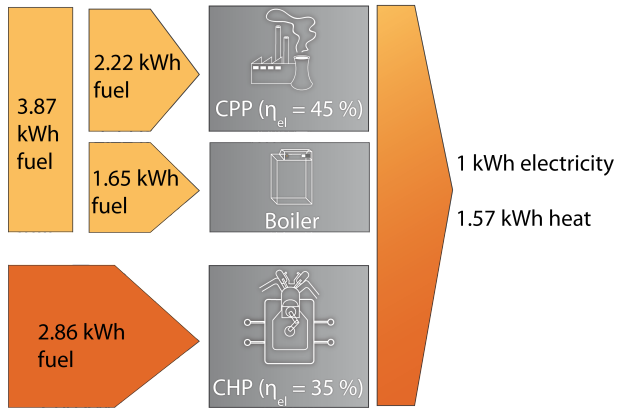


Figure 1: Comparison of combined heat and power generation and the separated generation [4]

the increasing dynamic of the system. Power plants are controlled with schedules that are generated from weather forecasts etc. The forecasts (one day before) and the real situation (e.g. solar radiation) can differ in a strong way (figure 2). This results in the need for fast reacting and fast starting and stopping technologies. For these applications, CHP units based on internal combustion engines are a suitable alternative because of their well controllable and fast reacting load conditions.

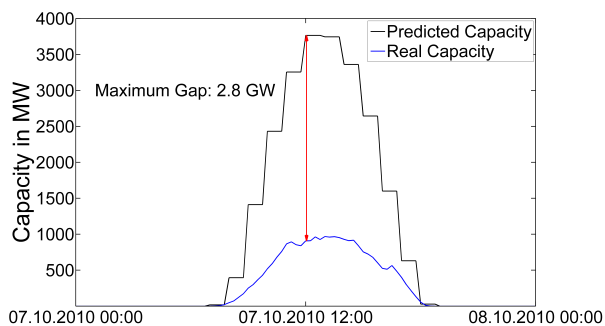


Figure 2: Comparison of predicted and real capacity of photovoltaics in the zone of one German grid operator [4]

For an efficient use of CHP units, the heat has to be used directly in the building. Residential buildings are predestinated as heat sinks for CHP plants, because of a year-round heat demand. This heat demand is signif-

icantly larger in winter because of the heating demand on colder days. In summer there is just the demand for domestic hot water. Due to this situation and the fact that the thermal demand does not coincide with the generation (especially with a power-controlled device), thermal storages play an important role in such a concept. Thermal storages are cheap in comparison to electrical storages with a similar capacity. The size and insulation of those thermal storages has to be analysed to find out which kind of storage should be used.

Besides the storage, also the single CHP units have to be analysed. Today, there are several micro-CHP units with capacities of 1 kW_{el} on the market for one-family-dwellings. Based on internal combustion engines, they deliver a thermal power of about 2.5 kW . If they are used in a power-controlled way, this could result in less delivered thermal power than would be needed for space-heating and domestic hot water supply. A solution to this problem could be an overdimensioning of the CHP plants, so that in times of a running plant, the thermal storage can be loaded very fast to provide a secure supply based on the CHP-plant as long as possible. For times, when a secure supply cannot be guaranteed with this stored thermal energy, a peak-load boiler should be installed.

Before those systems can be tested in reality in a power-controlled way, they should be analysed in simulations to get insight to most of the occurrent effects. Because of the complexity and dynamics of this system, a coupled thermal, hydraulic and rudimentary electrical simulation is used. Modelica with its equation-based modelling approach is a good tool to bring this complexity into a model based on single components. The used model will be explained in the following chapter.

3 Whole building system simulation

3.1 Design

For the evaluation of power-controlled micro-CHP-systems, a whole system model is needed. For the hydraulic components, the Modelica fluid-library is used. With this library, easy connection setups between single components of the model are possible. Standard components like pipes, vessels and valves can be composed to new components. It is also possible to connect the different elements to a whole hydraulic circuit. The used medium in this model is obviously water. All the components are interacting with each other, so that it is very difficult to just simulate one compo-

ment after the other. Especially the storage effects of the building mass and the included hot water storages can just be analysed in a coupled complete system simulation. The different models for simulation of power-controlled and heat-controlled operation are shown in figure 3.

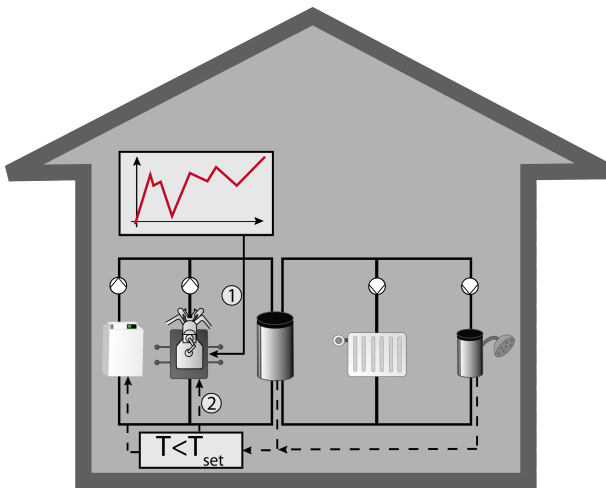


Figure 3: The whole building simulation model

The heat generating units are the CHP plant and the peak-load boiler. There are two storages included, one for space heating and one for domestic hot water. The boiler switches on if the stored heat cannot provide the heat supply in both heat-controlled and power-controlled operation. The CHP plant is either running based on a predefined profile (power-controlled, control 1 in figure 3) or running due to a temperature drop in the storages (heat-controlled, control 2 in figure 3). The heat transfer to the single rooms is guaranteed with radiators.

The model of the micro-CHP plant is built up on manufacturer's data [7]. It is mainly based on predefined table values. This means that a given relation between electrical power, thermal power and fuel consumption can be set in the model for different load types (part load in different steps and full load). This simple approach of modelling a CHP-unit gives us the possibility to study the behaviour of the total building energy system.

Some delay elements are included to improve the dynamics of start-up and shut-down processes. Those processes are very important to model a power-controlled CHP-unit. The model has an input to predefine the values forced by the electrical grid operator. This signal leads to a calculation of the belonging thermal power and fuel consumption. The calculated thermal power is fed to a volume element of the Modelica fluid library. If the plant runs heat controlled, it

just switches off or modulates if a predefined outflow temperature is reached. In both cases (heat-controlled and power-controlled), a superior control can be implemented to switch the plant on or off. This can be used for security applications etc.

As stated before, a buffer storage has to be integrated to decouple the generated heat from the demand. This decoupling is necessary both in the heat-controlled and in the power-controlled operation. The buffer storage is built up as a stratified storage with several layers which are thermally and hydraulically connected [6]. Besides the stored energy and the heat and mass transfer inside the storage, the heat losses to the environment (in the basement) have to be considered, because these losses will influence how long the stored energy can be used. The water elements inside the storage are volume elements of the Modelica fluid library. The fluid transfer between the layers is calculated automatically because of the connection to the hydraulic network of the building. The heat transfer between the layers is calculated with an approach of effective thermal conduction. For the heat transfer to the environment, a physical approach of heat transfer (convection, conduction, convection) in a tube is used. In the upper part of the domestic hot water storage, a heat exchanger is included to separate the heat supply circuit physically from the domestic hot water which has to stay very clean.

Besides the supply system, also the building has to be modelled. In the examined system, the house is modelled physically to represent all the storage and loss effects that can be observed in a house. These are for example all transmission losses due to the temperature difference between inside and outside. Besides, the storage capacity of the walls is also considered. If a wall consists of different layers (e.g. concrete and insulation material), those different layers are implemented with their storage capacity and heat transport properties. Another important effect are the ventilation heat losses of the building effected by infiltration and air exchange through natural ventilation caused by the user of the building. The user behaviour is also important for the thermal simulation of the building, because a human being produces heat itself and uses different electrical devices which produce additional heat.

The third thing where user behaviour plays a role is the domestic hot water tapping profile. The impact of the domestic hot water supply on the overall heat supply for buildings will increase as the space heat demand will gradually decrease in refurbished and new buildings. The supply of the users with domestic hot

water is implemented with a domestic hot water tank which is loaded by the CHP-unit and, if necessary, by the boiler. Another possibility to generate domestic hot water would be a fresh water station directly coupled to the buffer storage. This should be developed in the future.

To set the flow temperature of the space heating system to a desired value, a return addition is integrated. This element is especially interesting when there is demand for domestic hot water and space heating at the same time and the temperature level for the space heating is lower than the desired domestic hot water temperature. The building model includes heat valves that inhibit the fluid flow through the radiators if the desired indoor temperature is reached. The desired indoor temperature can be varied and is set from an input table. In the case described here, a temperature of 21°C during the day and 17°C during the night is set up. If the temperatures get higher than the desired value, the heat valves close and no fluid flows through the radiator anymore. As soon as the temperatures drop down, the valves open again and let the hot water pass. The power of the radiators is then calculated depending on the room temperature, the flow temperature, the surface area of the radiator, the nominal power and the radiator exponent.

3.2 First results

To show the possibilities of the model and the insights that can be obtained, two examples are shown. The first analysed plant has a maximum thermal power of about $2.5 kW_{th}$ and a maximum electrical power of $1 kW_{el}$. This is a standard micro-CHP unit which can be bought on the German market. Two models will be compared. On the one hand this is a model with a heat-controlled CHP-unit. This unit is only controlled by the temperatures in the buffer storage and the storage for domestic hot water. On the other hand, we analyse a power-controlled operation. In this second model, the times when the CHP unit runs are pre-determined by a certain profile that is set up maybe from a grid operator. Such a profile will result from a residual load profile. It is calculated for every time step as the difference between the electric load in the grid and the feed-in of renewable sources. If the residual load is above zero, some CHP units have to run because of a frequency drop in the grid. In times when the residual load is below zero, this energy has to be stored or maybe used in another way (in electricity-driven heat pumps for example). For two days, this profile is shown in figure 4.

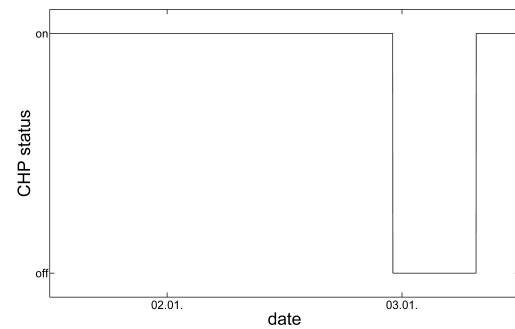


Figure 4: Desired on/off-profile of the CHP unit for two days

After running the simulation, the on/off-profile for the CHP-unit for two exemplary days is shown in figure 5 in the lower part. In contrast to that, the heat-controlled CHP-plant runs nearly continuously for all the days with a little exception at the beginning of January 2nd. At the same time, the power-controlled CHP-unit switches off, although the electricity-profile in figure 4 is set to on at this time. Both switch-off processes are due to an increasing temperature in the storage which is shown in figure 5 in the upper part. The storage volume is set to 1000 litres for both cases.

During the night, there is nearly no heat consumption which leads to an increasing temperature in the storage and also an increasing temperature of the fluid flowing to the CHP unit. To analyse these types of feedback between different parts of the energy system, a dynamic approach as it is used with Modelica is needed. We cannot fill the stratified storage until its whole temperature is at the maximum reachable temperature, because the plant has also some limits. These limits have to be analysed in further activities. To guarantee a secure energy supply, the plants in the single houses have to interact with each other. In a case when one plant cannot operate anymore due to an increasing temperature, this can be detected at an earlier stage and another plant can run instead.

The second example which is presented is a system with a bigger engine with an electrical power of $3 kW_{el}$ and a thermal power of $8 kW_{th}$. This plant is overdimensioned for the examined building, but we want to study how such a plant will behave in a heat supply system for one building. Due to the bigger thermal capacity of the plant, also a bigger thermal storage is included. This storage has a volume of 2500 litres. This storage volume is needed to guarantee longer run times of the plant. But, as we can see in figure 6, the storage capacity is not big enough to let the CHP-plant run

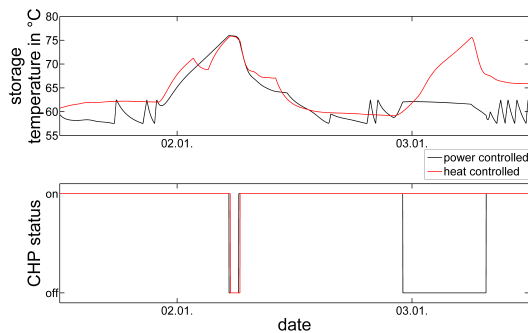


Figure 5: Temperature at the top of the buffer storage and the on/off-profile of the CHP unit

with the desired profile. The plant often runs in part-load instead of full-load or it is even switched off. This shows that a system like this can just work in a house with worse insulation or it has to share the production of the desired electrical power with another house.

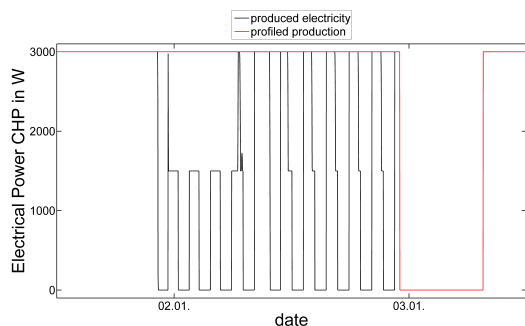


Figure 6: Desired profile and really produced electricity of the CHP unit (3 kW_{el}) for two days

Other effects like the behaviour of the peak-load-boiler can also be studied. In this example, the additional energy which should be provided by a peak-load-boiler is analysed. For the heat-controlled plant, no additional heat from the boiler is needed in the considered time period of two days. For the power-controlled plant with a capacity of 1 kW_{el} , an additional heat energy demand of about 44 kWh is calculated. This shows, that this operation will lead to higher demands for boilers. The solution for this could be the over-dimensioned power-controlled plant with a capacity of 3 kW_{el} . With this plant, just a little demand of about 3 kWh is calculated. With the approach of an over-dimensioned plant, we can decrease significantly the additional heat demand which is provided by a boiler. This does not regard the fact that the plant with 3 kW_{el} was not running in the pre-defined way. With a plant running as it was pre-defined, the heat

gained from the plant would increase significantly. If this operation would be shared to different houses as mentioned before, it would decrease again. Detailed analyses of these systems should follow.

In addition, the approach stated in this paper gives us the possibility to check, if the desired room temperatures are reached all the time with these new energy supply systems.

4 Conclusion and outlook

CHP plants can play a bigger role in the supply of residential buildings because of their flexibility and their energy efficiency. This paper shows, how Modelica can be used to model these new and distributed energy systems for the future. A model for the integration of the CHP plant in the heat supply for residential buildings is shown. The difference between heat-controlled and power-controlled operation modes is presented. Different user behaviour profiles can be included to improve the systems engineering adapted to the single user.

With this comprehensive model, a detailed analysis of future power-controlled micro-CHP-systems can be performed. Besides the analysis of power-controlled system, also switching between heat-controlled and power-controlled operation is possible and will be considered in the future.

Anyway, there are several things which should and will be implemented in the future. The model of the micro-CHP plants has to be validated with measurement data to be sure to represent the dynamics of the plant correctly.

The current models have to be simplified to less complex and less extensive models. This would provide the possibility to simulate more than one house in one model as it was mentioned in chapter 3.2. Thus, it would be possible to get the houses interconnected and to simulate the supply of whole city quarters with their electrical demand and fluctuating electricity sources like photovoltaics and wind power. This would give an integrated insight to the energetic impact of future energy systems with a higher rate of micro-CHPs.

A third point that has to be implemented is the interconnection of the different user profiles that are used in the model. In detail, these are the electrical demand profile, the domestic hot water profile, the natural ventilation profile and the heat source profile caused by internal loads. This would additionally improve the accuracy of the prediction for those energy systems.

References

- [1] EnEV 2009, Energieeinsparverordnung, Energy saving ordinance for buildings, 2009.
- [2] A. Hoh, T. Haase, T. Tschirner, D. Müller. *A combined thermo-hydraulic approach to simulation of active building components applying Modelica*. In Proc. of 4th International Modelica Conference, Hamburg, March 2005.
- [3] AG Energiebilanzen e.V., Anwendungsbilanzen für die Endenergiesektoren in Deutschland in den Jahren 2009 und 2010, 2011.
- [4] TenneT TSO GmbH, Actual and forecast photovoltaic energy feed-in, <http://www.tennetso.de/site/en/Transparency/publications/network-figures/actual-and-forecast-photovoltaic-energy-feed-in>, last call: 11.05.2012.
- [5] Aktualisierte und erweiterte Testreferenzjahre (TRY) von Deutschland für mittlere und extreme Witterungsverhältnisse, Bundesinstitut für Bau-, Stadt- und Raumforschung, 2011
- [6] K. Huchtemann, D. Müller, *Advanced simulation methods for heat pump systems* In Proc. of 7th International Modelica Conference, Como, Italy, September 2009
- [7] P. Jahangiri, Simulation and Comparison of Different District Heating Networks in Combination with Co-generation Plants, Master thesis, 2010

Modeling of a falling film evaporator

Alberto de la Calle^a Luis J. Yebra^a Sebastián Dormido^b

^aCIEMAT-Plataforma Solar de Almería, Ctra. de Senés s/n, 04200 Tabernas, Spain

^bUNED, Escuela Técnica Superior de Ingeniería Informática, 28040 Madrid, Spain

Abstract

Falling film evaporators have demonstrated a good performance in air-conditioning and refrigeration. This paper presents the development of a detailed falling film evaporator model. The model is based on classical Newton's viscosity law and Nusselt falling film theory. A library of evaporator components compatible with Modelica.Fluid, Modelica.Thermal and Modelica.Media has been implemented. The simulations presented have the expected behaviour. These models will be used to a complete model of a heat pump.

Keywords: Falling film; evaporator; heat pump; dryout; object-oriented modeling;

1 Introduction

One of the difficulties of working with solar energy is its variability. Since this technology starts, researchers have studied how to avoid solar irradiance disturbances affect energy production. The proposed solutions range from thermal storage to auxiliary energy sources to make feasible facilities.

With the aim of testing and developing a solar thermal Multi-Effect Distillation (MED), AQUASOL experimental thermal desalination plant was built at CIEMAT-Plataforma Solar de Almería at the early nineties [6]. Presently, the experimental plant performs an hybrid solar-gas process that combines, a thermal desalination system and a solar field with a Double Effect Absorption Heat Pump (DEAHP) coupled with a gas boiler [3] (Fig. 1). This system achieves at the same time the design requirements of low-cost, high efficiency and zero discharge [1].

The MED plant is a 14-effect plant where the seawater descends due to gravity from the 1st to 14th effects achieving a 3 m³/h nominal distillate production (Fig. 1). In the effect 1, the seawater is preheated by hot water (66.5 °C) coming from a 12-m³ primary storage tank. Energy supplied to the primary tank can be trans-

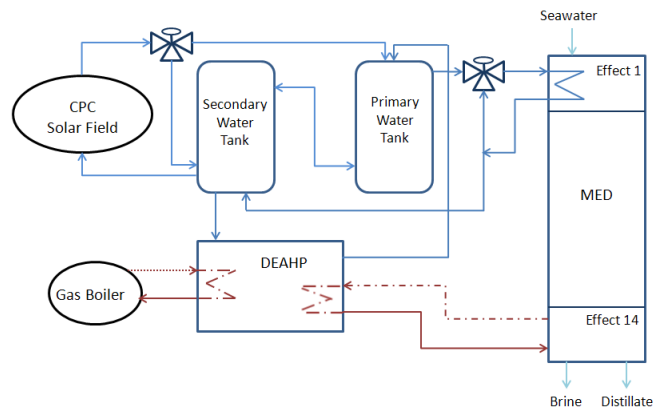


Figure 1: AQUASOL project plant flow sheet

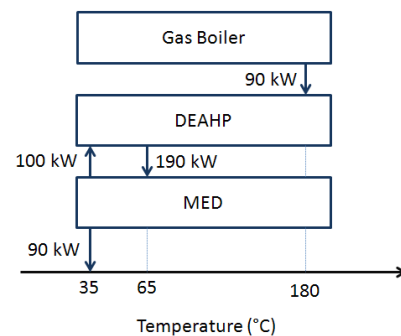


Figure 2: Energy balance of MED plant coupled to DEAHP

ferred by the solar field, by the DEAHP or by both. When the DEAHP is coupled with the MED plant, the DEAHP evaporator works as 14th effect distillate condenser.

A heat pump is a machine that transfers heat from a low temperature source to a high temperature source. In AQUASOL DEAHP, the low temperature source (35 °C) is the 14th effect cell and the high temperature source is the water that flows from the secondary tank (63.5 °C) to the primary tank (66.5 °C). According to the Second Law of Thermodynamics, an energy input is required to make this heat transfer possible. This energy is provided by steam generated at the gas

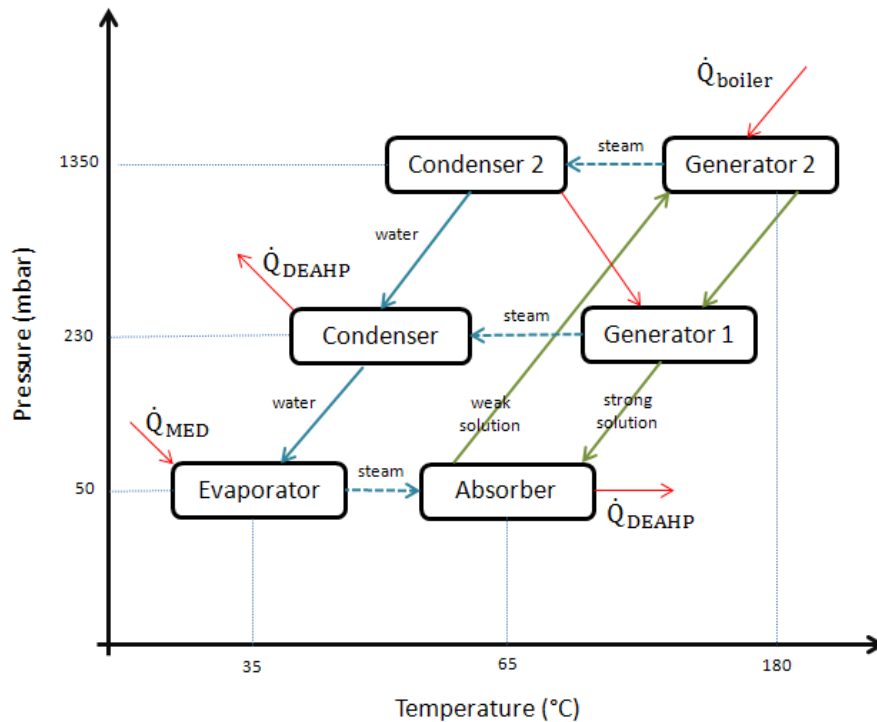


Figure 4: Thermodynamic cycle of the AQUASOL DEAHP



Figure 3: AQUASOL DEAHP, CIEMAT-Plataforma Solar de Almería

boiler (180 °C). Fig. 2 shows the heat transfer process between the MED plant, the DEAHP and the gas boiler [2].

AQUASOL DEAHP (Fig. 3) is composed of 5 vessels: one evaporator, one absorber, two generators and one condenser. It uses a water/aqueous lithium bromide solution as working fluid in two fluid interconnected circuits.

Fig. 4 shows the thermodynamic cycle of DEAHP. The low temperature source (steam from 14th effect cell) transfers heat to the low pressure evaporator which generates a steam flow. This steam is absorbed by a strong lithium bromide solution, as a consequence

it increases its temperature and transfers the heat to the high temperature source (water that flows from secondary to primary tank). To desorb the weak lithium bromide solution there are two generators with two different pressure levels. The gas boiler transfer heat to the generator 1 desorbing part of the water of the solution. The steam generated is condensed in generator 2 transferring heat and desorbing more water. The steam generated in generator 2 is condensed transferring heat to the hot source. Water from condenser and generator 2 returns to the evaporator and the strong lithium bromide solution return to absorber.

A detailed model of the DEAHP evaporator is presented in this paper. This study has been done under the framework of POWER project.

2 Mathematical model

The nomenclature used in this section is described in Appendix A. Newton's notation is used for time derivatives.

AQUASOL DEAHP evaporator is a horizontal-tubes-falling-film-type evaporator. Falling film evaporators have demonstrated better performance than flooded tubes evaporators in air conditioning and refrigeration applications due to its higher heat transfer

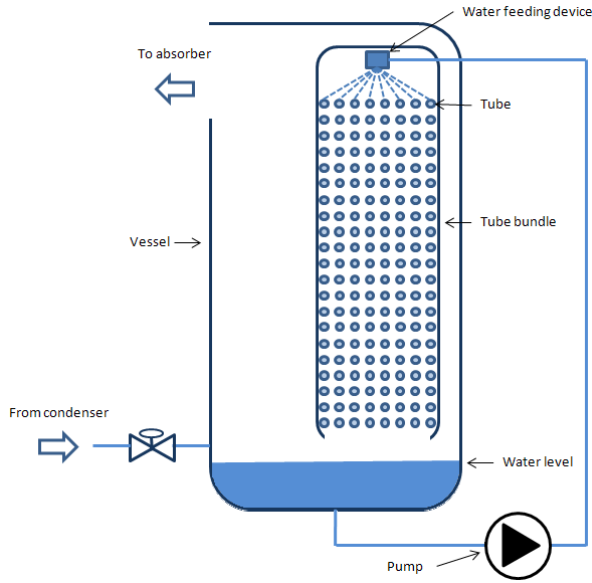


Figure 5: DEAHF evaporator scheme

coefficient and its smaller size [4].

A schematic cross section of the evaporator is shown in Fig. 5. Water is sprayed over the first row of the bundle tube structure. Over the tube surface, a thin film of water is formed. Water in the film flows downward under the gravitational force, falling one by one over all the column tubes. The film thickness determines the mass and heat flow rates.

The water feeding device can affect evaporator performance because it determines the water distribution over the tubes. In the model a uniform water distribution is assumed. When the water film flow rate falls below a certain limit dry patches are formed. This reduces the effective wetted area and consequently the heat transfer. An empirical correlation of the apparent wet area fraction $F = A_{wet}/A_t$ is presented in [5]. The fraction is calculated according to Eq. 1. If F is equal or less than 1, F is fixed to 1.

$$F = 0.0024Re_{top}^{0.91} \quad (1)$$

This correction is included in the model through the wetted length, l_{wet} , which is estimated with F and the real length l .

$$l_{wet} = lF \quad (2)$$

A classical formulation is used to model the dynamics of the falling film displacing over the tubes. It is assumed that the film thickness is constant over the tube as shows Fig. 6. According to it and using Newton's Law of Viscosity the force balance equation for the liquid film can be expressed as:

$$\mu \frac{du}{dy} = \sum \tau_s = \rho g(\delta - y) \sin(\theta) \quad (3)$$

Integrating over the spacial coordinates Eq. 3 and applying the boundary condition $u = 0$ at $y = \delta$, the velocity profile is:

$$u = \frac{\rho g}{\mu} \left(\delta y - \frac{y^2}{2} \right) \sin(\theta) \quad (4)$$

The downward average film velocity depending on the angle is:

$$\bar{u} = \frac{1}{\delta} \int_0^\delta \frac{\rho g}{\mu} \left(\delta y - \frac{y^2}{2} \right) \sin(\theta) dy = \frac{\rho g \delta^2}{3\mu} \sin(\theta) \quad (5)$$

The average film velocity over the tube is calculated integrating Eq. 5 over half of the tube circumference from the top to the bottom:

$$v = \frac{1}{\pi} \int_0^\pi \frac{\rho g \delta^2}{3\mu} \sin(\theta) d\theta = \frac{2g\rho\delta^2}{3\pi\mu} \quad (6)$$

The mass flow rate at the bottom of the tube is:

$$\dot{m}_{bot} = 2\Gamma l_{wet} = 2v\delta\rho l_{wet} = \frac{4gl_{wet}\rho^2\delta^2}{3\pi\mu} \quad (7)$$

Eq. 7 has a quadratic relationship with the film thickness.

The film thickness can be calculated using the density definition:

$$\rho = \frac{m}{V} = \frac{m}{\pi l_{wet}(r + \delta)^2 - \pi l_{wet}r^2}$$

$$\delta = -r + \sqrt{r^2 + \frac{m}{\pi\rho l_{wet}}} \quad (8)$$

where m , the mass of the water stuck to the outer surface of the tube, is calculated with the mass balance over the tube:

$$\dot{m} = \dot{m}_{top} - \dot{m}_{bot} - \dot{m}_{ev} \quad (9)$$

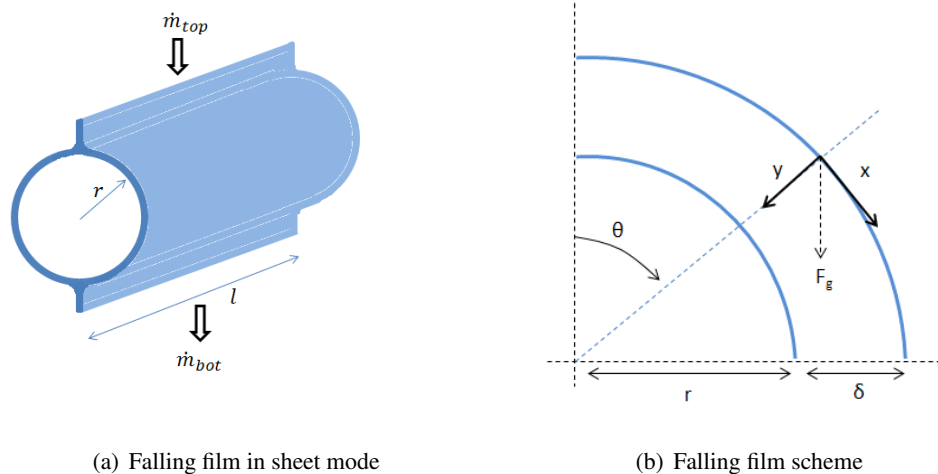
According to Nusselt's classical theory on falling film condensation, heat is transferred by conduction across the falling film. Same assumption is used in this model. Applying Fourier's law:

$$\dot{Q}_{tube} = kA_{wet} \frac{T_{tube} - T}{\delta} \quad (10)$$

The thin film energy balance is:

$$\dot{U} = \dot{Q}_{tube} + \dot{m}_{top}h_{top} - \dot{m}_{bot}h_{bot} - \dot{m}_{ev}h_{ev} \quad (11)$$

where \dot{Q}_{tube} is the heat flow rate transferred by the tube, \dot{m}_{bot} is calculated with Eq. 7, h_{bot} is assumed as the falling film specific enthalpy and h_{ev} the specific enthalpy of saturated vapor.



(a) Falling film in sheet mode

(b) Falling film scheme

Figure 6: Falling film on a horizontal tube

To determine the evaporated mass flow rate, the following relationship is used:

$$\dot{m}_{ev} = \frac{1}{h_{sg} - h_{sl}} (\dot{Q}_{tube} - \dot{m}h_{sl} - \dot{m}_{top}(h_{sl} - h_{top})) \quad (12)$$

where the mass of evaporated water is adapted with changes in the pressure and in the inlet mass flow rate.

The pressure inside the evaporator is estimated with the ideal gas law because the evaporator works at low pressures.

3 Modelica library

A new library to simulate a falling film evaporator has been developed using Modelica version 3.2. This library is completely compatible with Modelica.Fluid, Modelica.Thermal and Modelica.Media. Water thermodynamic properties have been calculated using the package Modelica.Media.Water.StandardWater.

The library is divided in the tube model, the tube conduction model, the tube column model and the tube bundle model.

The tube model is the basic class which includes all the equations that models the mass and energy balances in one single control volume. As inputs the model has a fluid port of top inlet flow and a heat port connected to the Nusselt falling film conduction model, as outputs it has two fluid ports, one for the outlet flow that falls by gravity at the bottom and one to evacuate the steam generated, and a real output to provide conduction model the heat transfer coefficient. The model obtains the water initial state variables (temperature and pressure) through an outer

Modelica.Fluid.System class. The parameters of the tube model are shown in Fig. 7.

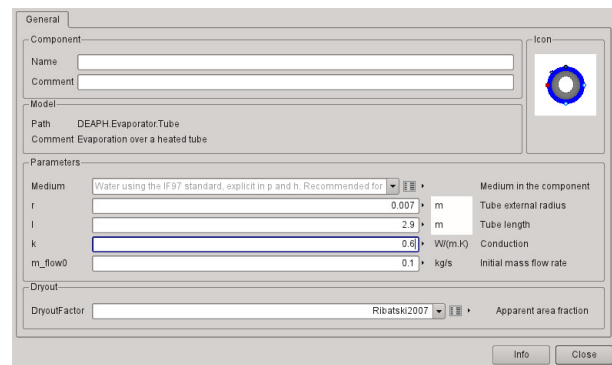


Figure 7: Tube model parameter menu

The tube conduction model joins a conduction model with a tube model, modeling the mass and energy dynamics of the falling film outside the tube. It has three fluid ports (top, bottom and steam) and one heat port where the metal tube transfers heat to the falling film.

The tube column model uses an array of tube conduction models in order to model one of the evaporator columns. The tube conduction models are connected consecutively one by one through top and bottom fluid ports, making a column of tubes. The steam fluid ports are interconnected between them in a single output fluid port. Also, the heat ports are interconnected between them, hence, assuming the same temperature. This assumption is possible because inside the tubes flow steam that is condensated transferring heat but keeping constant the temperature.

The tube bundle model adds to the tube col-

umn model the dynamics of an evaporator with many columns. This model assumes that all the columns have the same dynamics and extrapolates the results of one single column to all of them. The model additionally includes a conduction model where the conduction across the tubes is modeled as one single mass that transfers heat from the single mass to the outside.

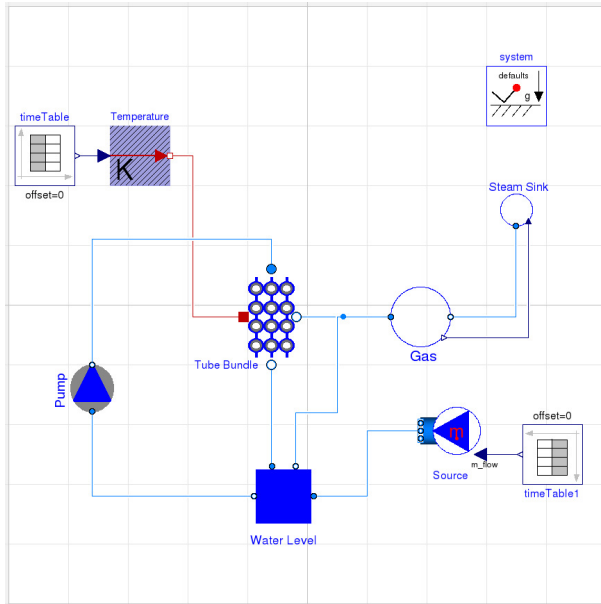


Figure 8: DEAHF evaporator in Modelica

4 Simulation

The model used to simulate the evaporator shown in Fig. 5 is composed of a tube bundle model, a water level model, a pump model, a gas model, a steam sink model and a water source model. The water level model simulates the mass and energy balances of the vessel under the tubes. The water in the vessel is recirculated with the help of a pump to the tube bundle as it is shown in Fig. 8. The gas model considers the mass and energy balances of a mixture of gases (steam and air) in the evaporator vessel where the pressure is given with the ideal gas law. The steam sink model is a first approximation to model the absorber where the steam leaves the vessel always with a steam mass flow rate lower than the saturation boundary. Water source model is used to control the vessel water level. The simulated tube bundle has 8 columns and 25 rows.

The simulation time was 25.8 s of 100 s. It has been performed starting from steady-state initial conditions. When the evaporator is started, an ideal pump flows a constant water flow rate to the tube bundle and each tube in the model has the same inlet flow rate. Water

and tubes start at the same temperature. At simulation time 20 s, the tubes progressively raise its temperature until time 40 s and where its temperature reach steady-state. Water starts evaporating. First, water heats the gas increasing slightly the pressure, and then, the gas leaves the vessel. When the steam sink model reaches its saturation mass flow rate, the pressure inside the vessel raises and that increases also the specific enthalpy of saturated liquid. This pressure rise stops when the evaporated mass flow rate reaches the saturation mass flow rate boundary given by the steam sink model.

Simulation results are shown in Fig. 9. Fig. 9a at the top depicts the total heat flow rate transferred by the tubes to the falling film. At the bottom a comparison between tube wall temperature and inlet and outlet water temperature of the tube bundle is shown. Fig. 9bI shows the variation of the pressure inside the vessel. This variation affects to the specific enthalpy of the saturated liquid as is depicted at Fig. 9bII. Also, in this figure is shown that while the 25th tube row begins to evaporate, at the 1st tube row the specific enthalpy of the inlet water is lower than the specific enthalpy of saturated water until the evaporator heats all the water in the vessel and the water in the vessel reaches the saturated temperature. Evolution of dry patches in tubes can be observed in Fig. 9c. Dryout disappears when water temperature increases and the Reynolds number increases too. Besides, in this figure can be observed the evolution of the falling film thickness along the experiment. Fig. 9d shows the steam mass flow rate generated in the evaporator. As it can be seen, the simulation has chattering in some of the tubes when the pressure increases, even though this effect has been foreseen in Eq. 12. Numerical errors taken into account could probably be the origin of this problem.

5 Conclusions

A new dynamic model of a falling film evaporator has been developed. The model is framed in a project which studies the AQUASOL DEAHF. A Modelica library for falling films evaporators has been implemented. The library is based on Newton's viscosity law and Nusselt's classical theory of falling film and it is compatible with Modelica.Fluid, Modelica.Thermal and Modelica.Media libraries. The simulations show the expected performance within the range which it has been designed in spite of chattering in evaporation. The chattering problem will be studied in detail in future works and possible solutions like hysteresis

are going to be tested. The library will be extended with new components that will model absorbers and generators. Models will be calibrated and validated with experimental data and control algorithms will be proposed to optimize the DEAHF performance.

Appendix A. Nomenclature

A	Area (m^2)
F	Apparent wet area fraction (dimensionless)
g	Gravitational acceleration ($\text{m} \cdot \text{s}^{-2}$)
h	Specific enthalpy ($\text{J} \cdot \text{Kg}^{-1}$)
l	Length (m)
m	Mass (Kg)
k	Conductivity ($\text{W} \cdot \text{m}^{-1} \cdot \text{K}^{-1}$)
p	Pressure (Pa)
Q	Heat (J)
Re	Reynolds number $4\Gamma/\mu$ (dimensionless)
r	Radius (m)
T	Temperature (K)
U	Internal energy (J)
u	Flow velocity ($\text{m} \cdot \text{s}^{-1}$)
V	Volume (m^3)
v	Average film velocity ($\text{m} \cdot \text{s}^{-1}$)
x	Spatial coordinate tangential to the tube (m)
y	Spatial coordinate normal to the tube (m)

Greek symbols

δ	Film thickness (m)
Γ	Liquid mass flow rate per unit length of tube (each side) ($\text{Kg} \cdot \text{m}^{-1} \cdot \text{s}^{-1}$)
θ	Angle (rad)
μ	Dynamic viscosity ($\text{Kg} \cdot \text{m}^{-1} \cdot \text{s}^{-1}$)
ρ	Density ($\text{Kg} \cdot \text{m}^{-3}$)
τ_s	Shear stresses (Pa)

Subscripts

bot	bottom
ev	evaporated
sg	saturated gas
sl	saturated liquid
t	total
top	top
tube	tube
wet	wetted

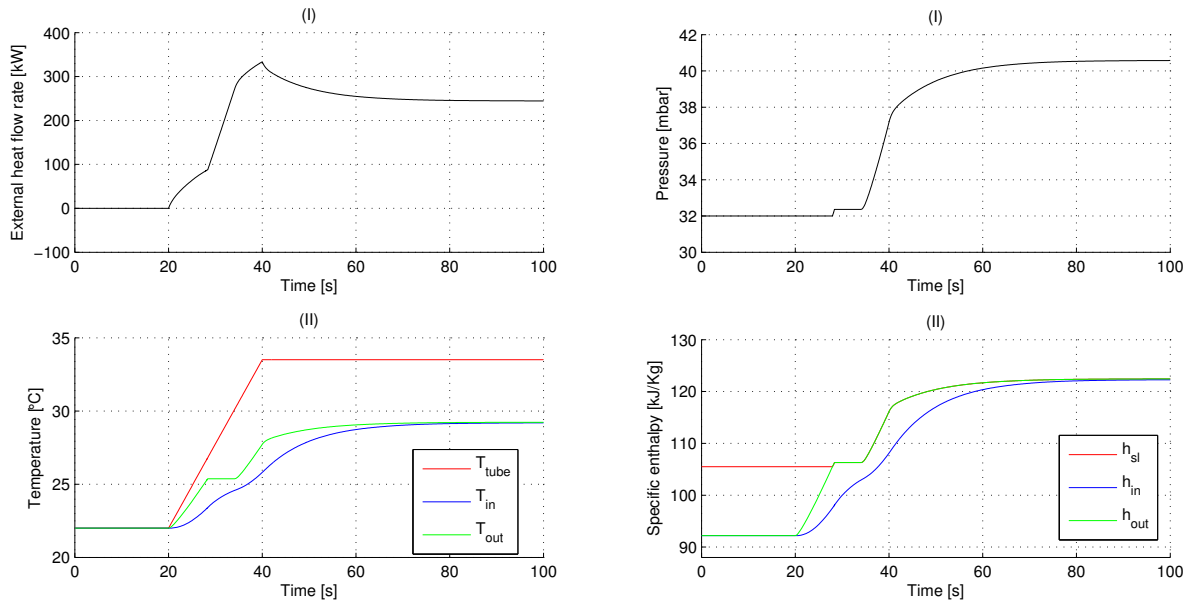
Acknowledgements

The authors would like to thank to CIEMAT research centre, Spanish Ministry of Economy and Competitiveness and FEDER funds for financed this work un-

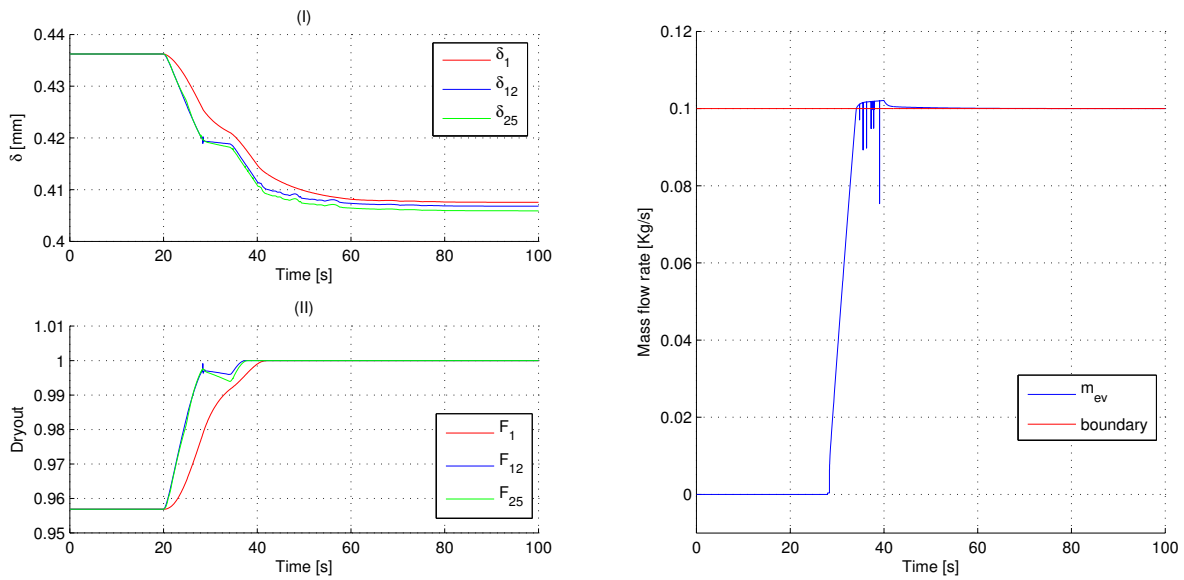
der the National Plan Project, Predictive control techniques for efficient management of renewable energy microgrids. (POWER), DPI2010-21589-C05-02 and the INNFACTO project, Hibridación de tecnologías renovables en una planta de generación de energía. (HIBIOSOLEO), IPT-440000-2010-004.

References

- [1] D. C. Alarcón-Padilla, J. Blanco-Gálvez, L. García-Rodríguez, W. Gernjak, and S. Malato. First experimental results of a new hybrid solar/gas multi-effect distillation system: the AQUASOL project. *Desalination*, 220(1-3):619–625, 2008.
- [2] D. C. Alarcón-Padilla, L. García-Rodríguez, and J. Blanco-Gálvez. Assessment of an absorption heat pump coupled to a multi-effect distillation unit within AQUASOL project. *Desalination*, 212(1-3):303–310, 2007.
- [3] D. C. Alarcón-Padilla, L. García-Rodríguez, and J. Blanco-Gálvez. Experimental assessment of connection of an absorption heat pump to a multi-effect distillation unit. *Desalination*, 250(2):500–505, 2010.
- [4] G. Ribatski and A. M. Jacobi. Falling-film evaporation on horizontal tubes—a critical review. *International Journal of Refrigeration*, 28(5):635–653, 2005.
- [5] G. Ribatski and J. R. Thome. Experimental study on the onset of local dryout in an evaporating falling film on horizontal plain tubes. *Experimental Thermal and Fluid Science*, 31(6):483–493, 2007.
- [6] E. Zarza, J. Ajona, J. León, A. Gregorzewski, and K. Genthner. Solar thermal desalination project at the Plataforma Solar de Almería. *Solar Energy Materials*, 24(1–4):608–622, 1991.



(a) (I) Inlet heat flow rate across tubes (II) Tube wall temperature, (b) (I) Pressure in the evaporator (II) Specific enthalpy of saturated liquid, inlet water specific enthalpy, outlet water specific enthalpy



(c) (I) Falling film thickness in tube 1, 12 and 25 (II) Apparent wet area fraction (d) Evaporated water mass flow rate and sink saturation boundary

Figure 9: Simulation results

Integration of Modelica models into an existing simulation software using FMI for Co-Simulation

Matthias Pazold¹, Sebastian Burhenne², Jan Radon³, Sebastian Herkel² and Florian Antretter¹

¹Fraunhofer Institute for Building Physics, Holzkirchen, Germany

²Fraunhofer Institute for Solar Energy Systems, Freiburg, Germany

³Agr. University of Cracow, Poland

Matthias.Pazold@ibp.fraunhofer.de

Abstract

The Functional Mock-up Interface (FMI) opens new opportunities for the development and extension of existing non-Modelica simulation programs with Modelica models. For the developer this is a productive way to design and validate new complex simulation models with multi-domain modeling languages such as Modelica. With the standardized Functional Mock-up Interface (FMI) and the Functional Mock-up Unit (FMU) export it is possible to execute these models within other software tools, including information exchange during the simulation. However, there are some design requirements in Modelica, which have to be taken into account. In this paper, models for different HVAC (Heating, Ventilation and Air Conditioning) equipment configurations are integrated into existing software using the FMI. An interface extension plug-in is developed to pick a specific FMU and execute it alongside the existing simulation algorithm. Two different coupling algorithms were investigated: the iterative and the co-simulation approach. Some issues and practical hints for a successful coupling and simulation are presented.

Keywords: Building Simulation; FMI for Co-Simulation; HVAC

1 Introduction

The application of building performance software during the design process is standard in the design of energy efficient buildings. There are tools that solve the coupled heat and moisture transport in building components to avoid moisture related problems such as mold growth or rotting components. Different kinds of components (e.g. walls, windows, roof) are combined to a whole building model. Additionally,

climate data and inner sources lead to a whole building envelope simulation software, which allows for an accurate assessment of the indoor environment and the energy consumption of the building. The WUFI[®]Plus software offers the possibility for such a simulation. Until now, the HVAC equipment was considered as an ideal heating and cooling system. Current activities aim to implement realistic models into WUFI[®]Plus to simulate HVAC systems. These models are written in Modelica [1]. The building envelope and the HVAC system influence each other significantly. This makes a separate simulation of both systems inaccurate and introduces special requirements for combining both in a co-simulation. The decision to implement the Modelica models into the existing software rather than model the building envelope with Modelica was made because of the big user community, which is familiar with the existing GUI and other user specific requirements. A possible way to include Modelica models into an existing building simulation program is the Functional Mock-up Interface for Co-Simulation. The integration is described in this paper.

2 Existing Software / Models

2.1 Building model

WUFI[®]Plus is a holistic model based on the hygrothermal envelope calculation model developed by Künzel [2]. The hygrothermal behavior of the building envelope affects the overall performance of a building. WUFI[®]Plus is a building performance simulation tool, which computes the coupled heat and moisture transfer in the building components. These components are combined to a whole building model. Moisture sources or sinks inside the rooms or components, input from the envelope due to capillary

action, diffusion and vapor ab- and desorption as a response to the exterior and interior climate conditions as well as the thermal parameters are taken into account. A stable and efficient numerical solver had been designed for the solution of the coupled and highly nonlinear equations. The conductive heat and enthalpy flow by vapor diffusion with phase changes in the energy equation are strongly dependent on the moisture fields. The vapor flow is simultaneously governed by the temperature and moisture field due to the exponential changes of the saturation vapor pressure with temperature. The differential equations are discretized by means of an implicit finite volume method. The model was validated by comparing its simulation results with the measured data of extensive field experiments [3]. The user can define design conditions for the indoor climate by setting minimal and maximal values. To simulate the indoor climate, the software calculates heat and moisture balances for one or more building zones, regarding all the sources, sinks and transfers. So long as these balances are not satisfied during a time step, the interior temperature and humidity is adapted. For example, if the heat loss through the building envelope and ventilation is more than the solar and internal heat gains plus space heating capability, the interior temperature is decreased as long as the loss and the gain is equal.

2.2 Modelica HVAC models

The aim was to create simple but realistic HVAC models, which can be used by practitioners. This means that only necessary and obtainable plant information is required for these simulations. The computation time to simulate a building should not increase to times which are no longer acceptable for practitioners.

Systems to be simulated include:

- Condensing gas boiler
- Solar thermal collector
- Combined heat and power plants
- Heat pumps
- Bore hole heat exchangers
- Thermally activated building systems (TABS)
- Radiators
- Storage tanks
- Control equipment
- PV systems

The model development was done with the software Dymola 2012 [4]. To deliver realistic and validated plant equipment models, the above mentioned sub-models are merged to complete HVAC configurations, an example is shown in Figure 1. This was done to increase the usability and avoid the risk of non-feasible configurations. In the end, the user chooses one HVAC configuration and has to select only a few necessary parameters or import them from a database.

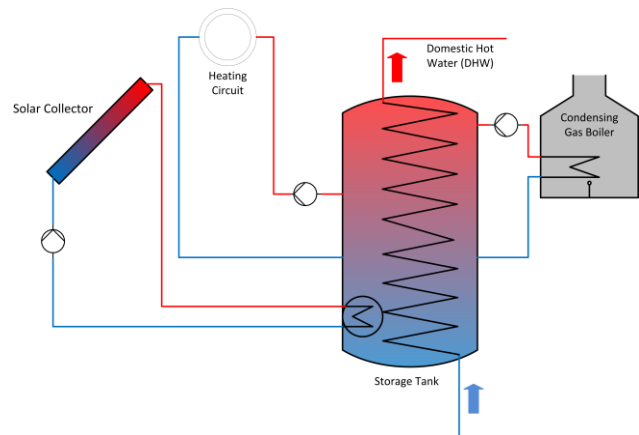


Figure 1: Exemplary HVAC configuration in WUFI®Plus

3 Integration

The first investigated coupling approach was to use Dymola specific export possibilities (*Source Code generation*). More details on this can be found in [5]. Finally the coupling with the Functional Mock-up Interface for Co-Simulation was chosen because of its unified convention and possibilities to perform the co-simulation. Merging the existing software and the Modelica HVAC models using the Functional Mock-up Interface for Model Exchange would require the development of a new solver for WUFI®Plus. Therefore, one of the main advantages in the context of the described application was that a solver is included in the FMU for Co-Simulation. In the described case it is the CVODE solver included in the Sundials solver package [6]. The selected solver within Dymola has no influence on the exported solver. The standardized interface provides some methods to interact with the model. Beside instantiating, initializing, setting and obtaining of values of defined variables and parameters there is the possibility to execute single time steps. Furthermore, there is a distinction between time varying variables and parameters. The value of parameters can be set before initializing the model; the value of variables can be set between the time steps. But these time

varying variables must be declared as input within the Modelica code. Their causality must be set to *Input*. If parameters appear in if-statements in sub-models, the model must be re-compiled for a change of their value. In the exported and compiled FMU such a parameter is automatically set to constant and the value is firmly anchored. Changing the value of constants within the FMU is not possible. If-statements are often responsible for discontinuities and events. They should be avoided during the model design process because they increase the computation time [7]. However, to set a parameter of an if-statement in the compiled FMU, a workaround is to define the parameter as input.

There are more than one HVAC configurations with different devices and different parameters and, consequently, many FMUs. WUFI®Plus has to interact with the HVAC system configuration, which is chosen by the user of the software. A FMU adapter (Figure 2) is written in the object-oriented language C++ to manage dynamic FMU instantiation, initialization, set inputs, obtain outputs and execute time steps. Therefore, the adapter receives information about the different kinds of configurations and their parameters (their value references).

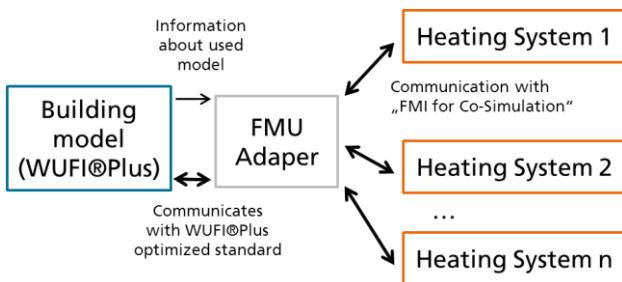


Figure 2: Communication between building model and heating systems

As mentioned, the building model and the HVAC models have to interact with each other. Some results of one are needed as input for the other. Two different insertion algorithms were investigated and are discussed below.

3.1 Iterative approach

As described before, WUFI®Plus uses an iterative process to simulate the interior temperature and moisture for defined zones. Also airflow is calculated iteratively. For short computation times there is a solver designed for fast convergence of these values with only a few iterations. Indeed, the HVAC systems influence the indoor climate. The first approach was to use the existing heat and moisture balance

algorithm. The HVAC system receives, for example, the indoor set point temperature and the actual temperature of a zone and a time step and delivers the possible heat flow to the zone. If the heat balance is not satisfied, the current temperature will be in- or decreased and the HVAC system must iterate (Figure 3).

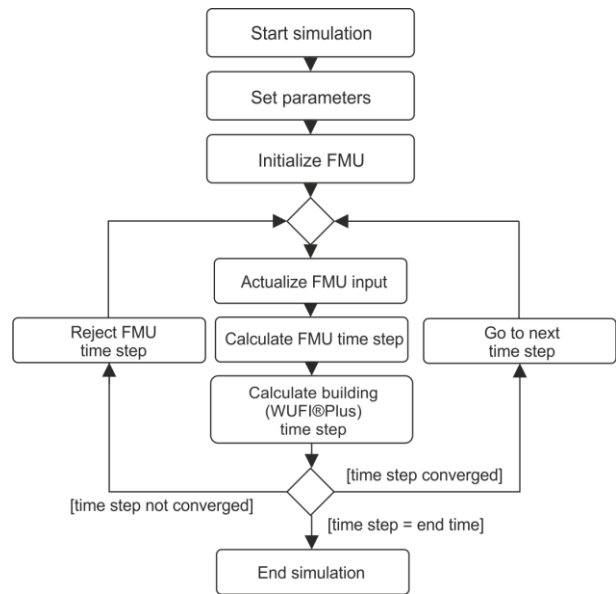


Figure 3: Flow chart - iterative approach

The advantage of this approach is to use the established flexible balance system. The HVAC model can be coupled in a fast way with only a few modifications of the WUFI®Plus algorithm. However, this method requires repeating and discarding of FMU time steps. Therefore the parameter `newStep` of `fmiDoStep(...)` can be set to `fmiFalse` if the capability flag `canRejectSteps` of the FMU is true. Until now this feature is not supported by the exported FMU. This is specific to Dymola and might not be the case for other simulation environments. However, in the analyzed case the missing feature is a problem for the implementation of the iterative approach.

If a time step is regarded as an entire simulation, a workaround could be to re-initialize the FMU for every time and iteration step. In order to retain all information, all time varying variables must be stored after a step and re-stored as initialization values for the next step. To repeat a step, the values of the last step are used. This could be time and memory consuming. Furthermore, some states of the model, which cannot be stored in the cache, may change during a time step.

A further issue of this coupling approach is that the iteration might end in a continuous loop. The heat supply system models are designed to deliver a heat

flow to the room, when the current room temperature is lower than the set point temperature. This is implemented using a thermostatic valve model. The building model iterates the room temperature with the heat balance, including the heating system as a kind of heat source. If the balance is positive, the room temperature can be increased for the next iteration step. In this case, the current room temperature might become equal or greater than the set point temperature and in return the heating system model calculates no heat gain. If the heat balance ends with a negative sum, the room temperature is decreased. The heat supply system reconvenes a heat gain for the next iteration. This leads to a continuous loop.

3.2 Co-simulation approach

The mentioned issues with the iterative approach lead to a real co-simulation approach. The iterative process has been removed, so there is no requirement to repeat time steps within the FMU. Therefore the building envelope model (WUFI®Plus) and the HVAC model calculate the steps alternately with a *ping-pong* method. A usual simulation time step, to simulate a whole year, is one hour. For the alternately co-simulation this time step size, with .e.g. constant room temperatures, very likely leads to unrealistic simulation results. One physically realistic solution is to decrease the time step size.

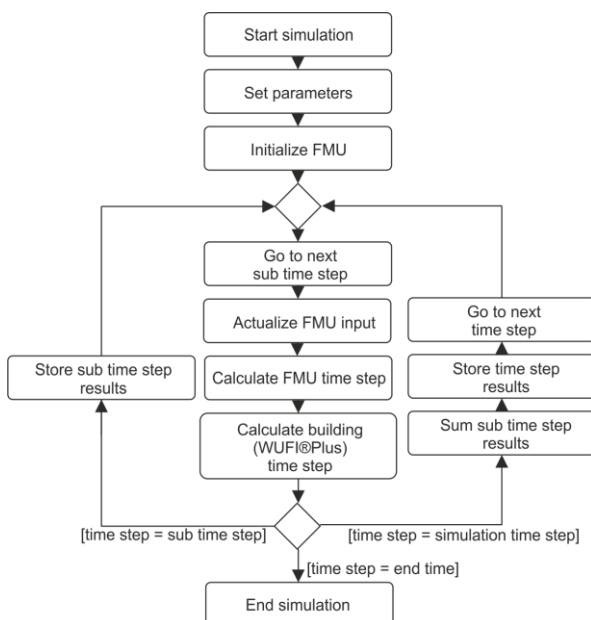


Figure 4: Flow chart - co-simulation approach

The explicit algorithm (Figure 4) is the following, described with thermal values: The plant equipment model calculates a few seconds with a constant interior temperature. Dependent on the heat emitting

system (e.g. radiator) and its heat capacity and performance, this is possible because of the fast response time of the active HVAC system. The result is still the heat flow which is added as heat gain to the building zones. Simultaneously, the building is simulated with the last heat gain. Depending on the heat balance the new interior temperatures for the next time step are calculated. With this method, small time steps depending on the time constant of the heating system must be used (e.g., five seconds). This leads to increased computation times. However, these small sub time steps must not be stored in the results. Furthermore, the building model converges faster with small time steps, which saves some computation time.

First tests with a time step of five seconds showed an increased demand of simulation time of about one third compared to the WUFI®Plus simulation without the HVAC models. The results of a Dymola simulation compared to the results of an external FMU simulation of the HVAC system are equal.

4 Conclusions

The multi-zone building model and the HVAC models are complex models with a lot of variables and their own specialized solver. Separately they are proven, validated and stable for many kinds of simulations. The described *weak coupling* using the co-simulation approach seems to be a reasonable technique. Exported FMUs, acting as sub-models with defined in- and output, can be used to supplement the building model. The authors believe, that in principle, the merging of the models is possible with the introduced iterative and co-simulation approach. However, not supported features of the exported FMU make the iterative approach unfeasible. A still acceptable computation time with the co-simulation approach led us to the conclusion that this is a more suitable approach in the described case. The sum of heat gains over the sub time steps delivered by the HVAC configuration is realistic. However, future work will include more investigations about the chosen time intervals and the handling of discontinuous input.

Acknowledgement

This study was funded by the German Federal Ministry of Economics and Technology (BMWi 0329663L)

References

- [1] Elmqvist, Hilding 1997. "Modelica – A unified object- oriented language for physical systems modeling." *Simulation Practice and Theory* 5, no. 6., 1997.
- [2] Künzel, H. M. 1994. *Simultaneous Heat and Moisture Transport in Building Components*. Dissertation. University of Stuttgart, Download: www.building-physics.com
- [3] Lengsfeld, K.; Holm, A. 2007. Entwicklung und Validierung einer hygrothermischen Raumklima-Simulationssoftware WUFI®-Plus, *Bauphysik* 29 (2007), Heft 3, Ernst & Sohn Verlag für Architektur und technische Wissenschaften GmbH & Co. KG, Berlin.
- [4] Dassault Systèmes AB 2011. *Dymola. Dynamic Modeling Laboratory. Dymola Release notes*, Lund, Sweden.
- [5] Burhenne, S.; Radon, J.; Pazold, M.; Herkel, S.; Antretter, F. 2011. Integration of HVAC Models into a Hygrothermal Whole Building Simulation Tool, *Proceedings of Building Simulation 2011: 12th Conference of International Building Performance Simulation Association*, Sydney, Australia.
- [6] Hindmarsh A. C.; Brown P. N.; Grant K. E.; Lee S. L.; Serban R.; Shumaker D. E.; Woodward C. S. 2005. SUNDIALS: Suite of Nonlinear and Differential/Algebraic Equation Solvers, *ACM Transactions on Mathematical Software*, 31(3), pp. 363-396, 2005. Also available as LLNL technical report UCRL-JP-200037.
- [7] Felgner, F.; Liu, L.; Frey G. June 2011. Vergleich numerischer Löser zur Simulation steifer und hybrider Systeme. *Proceedings of the Kongress Automation 2011, VDI-Berichte 2143*, Baden-Baden, Germany, pp. 303-306 (extended 12-pages paper on CD), June 2011.

Chemical Process Modeling in Modelica

Ali Baharev Arnold Neumaier
Fakultät für Mathematik, Universität Wien
Nordbergstraße 15, A-1090 Wien, Austria

Abstract

Chemical process models are highly structured. Information on how the hierarchical components are connected helps to solve the model efficiently. Our ultimate goal is to develop structure-driven optimization methods for solving nonlinear programming problems (NLP). The structural information retrieved from the JModelica environment will play an important role in the development of our novel optimization methods. Foundations of a Modelica library for general-purpose chemical process modeling have been built. Multiple steady-states in ideal two-product distillation were computed as a proof of concept. The Modelica source code is available at the project homepage. The issues encountered during modeling may be valuable to the Modelica language designers.

Keywords: separation, distillation column, tearing methods, homotopy continuation, bifurcation

1 Introduction

The object-oriented component-based modeling methodology in Modelica (FRITZSON [13]) is well-suited for chemical processes modeling. Accordingly, Modelica has received attention in the chemical engineering literature (SANDROCK & DE VAAL [19]). Creating a component-based framework for chemical process modeling is one of the goals of our project. This framework then serves as a common language between mathematicians and chemical engineers. The current chemical engineering literature is hardly accessible to mathematicians, partly due to the engineering jargon and unwritten traditions.

We created a prototype Modelica implementation of basic chemical engineering processes. Currently, only steady-state models are supported. Once this component library is finished, software with a graphical user interface, such as the [OpenModelica Connection Editor](#) (OMEdit), can be used to build chemical process models. The process

model creation involves only high-level operations on a GUI; low-level coding is not required. This is the desired way of input. Not surprisingly, this is also how it is implemented in commercial chemical process simulators such as [Aspen Plus](#)[®], [Aspen HYSYS](#)[®] or [CHEMCAD](#)[®].

Nonlinear system of equations are generally solved using optimization techniques. AMPL (FOURER et al. [12]) is the de facto standard for model representation and exchange in the optimization community. Many solvers for solving nonlinear programming (NLP) problems are interfaced with the AMPL environment. We are aiming to create a ‘Modelica to AMPL’ converter. One could use the Modelica toolchain to create the models conveniently on a GUI. After exporting the Modelica model in AMPL format, the already existing software environments (solvers with AMPL interface, AMPL scripts) can be used. Thus an AMPL export facility builds a bridge between Modelica users and the optimization community. Such an implementation exists (ÅKESSON [3]) but it is no longer supported, and not publicly available.

Our ultimate goal is the development of structure-driven optimization methods for solving nonlinear programming problems (NLP). The structural information (hierarchical components and the connections between them) can be exploited to solve the underlying process model efficiently. For example the process model of the reactive distillation column in CIRIC & MIAO [8], producing ethylene glycol from ethylene oxide and water, has 70 variables and 70 equations. However, the steady-state process model can be solved by solving univariate equations only, in a proper elimination order (BAHAREV & NEUMAIER [5]). In other words, the problem is essentially 1-dimensional. Typically, chemical process models are essentially low-dimensional even if their steady-state model is large-scale.

The structural information is difficult to get from an AMPL source directly, one would rather try to extract it from the flattened AMPL file instead. In prin-

principle, one could recover the required structural information from the flattened model, at least to some extent. This means that the flattening step throws away the structural information first, then one must try to recover it inside a solver. In contrast, the structural information is programmatically accessible in JModelica (ÅKESSON et al. [2]) before flattening, and we intend to utilize this.

2 Component-based modeling of chemical processes

Chemical processes are well-suited for component-based modeling since they are networks of equipments. In turn, it is natural to model the equipments hierarchically, as a composite of smaller components. The smallest subcomponents are called **atomic units**. The atomic units are connected by **process streams**.

2.1 Connector class: process streams

A process stream S consisting of C substances has $C + 2$ independent variables. It is characterized by the list of variables

$$S = \{S.f, S.p, S.H\},$$

where $S.f$ is an array of size C . See also Table 1.

variable	physical meaning	SI unit
$f[i] \geq 0$	molar flow rate of substance $i = 1 : C$	mol/s
$p \geq 0$	pressure	Pa
H	enthalpy flowrate	J/s

Table 1: The $C + 2$ variables characterizing a process stream.

The graphical representation of process streams is by arrows, as shown in Figure 1.

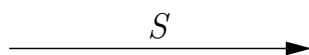


Figure 1: The graphical representation of stream S .

The units are connected by streams. The streams entering the unit are called **inlets**, while the streams leaving it are called **outlets**. The causal flows reflect the fact that the chemical process streams are directed, the material can only flow into the direction specified.

2.2 Sources and sinks

Given their simplicity, the easiest way to describe these components is by their implementation, see below. The only equations that sources and sinks can be

involved in are the connecting equations and specifications on their stream variables.

```
class Source
  output Stream outlet;
end Source;

class Sink
  input Stream inlet;
end Sink;
```

2.3 Types of equations

These equations apply to all atomic units in subsection 2.4. Only flows of chemicals are considered. Heat flows allowing thermal coupling or multidomain models would need an extension.

Material balances: A system of C linear equations, reflecting the conservation of mass.

Heat balance: A linear equation reflecting the conservation of energy.

Mechanical equilibrium: The outlets have the same pressure as the unit. With the exception of the mixer and the pressure changer, the pressure of the unit equals the pressure of its only inlet.

Thermal equilibrium: The enthalpy of the outlets corresponds to the temperature of the unit. This relation is expressed by nonlinear equations (equation of state). If the temperature is not an internal variable of the unit then these nonlinear equations are missing.

Characterizing equations: These equations characterize how the unit works and cannot be changed.

Connections with other units: These equations describe how the units are connected by equating the corresponding variables of the involved streams.

Specifications: These equations make the steady state model of the unit well-defined. They usually correspond to closed loop control systems. The form of these equations shows large variation: they can be trivial equations as well as complicated nonlinear equations.

2.4 Atomic units

As the name suggests, these units cannot be decomposed further to smaller, connected Modelica components. Atomic units implement the `UnitOp` interface, that is all the equations listed in Subsection 2.3 apply. These units are the followings.

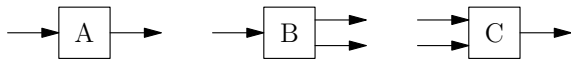


Figure 2: Structural types of the atomic units: (A) heat exchanger, pressure changer, reactor; (B) divider, flash; (C) mixer.

1. Mixer
2. Heat exchanger
3. Pressure changer
4. Reactor
5. Divider
6. Flash

The mixer has multiple inlets and a single outlet. All other atomic units have a single inlet and can have either one or two outlets. See Figure 2. Some code snippets are shown below. The simplicity of the implementation is a consequence of proper decomposition.

```
class Mixer
  extends UnitOp(nInlet=2, nOutlet=1);
end Mixer;

class PressureChanger
  extends UnitOp(nInlet=1, nOutlet=1);
  redeclare class ChangeInPressure=DeltaP;
end PressureChanger;

class Divider
  extends UnitOp(nInlet=1, nOutlet=2);
  Real zeta;
  equation
    outlet[1].f = zeta*inlet[1].f;
    outlet[1].H = zeta*inlet[1].H;
end Divider;
```

The Divider has one so-called unit parameter, ζ , its value typically comes from specification.

The atomic units or the equipments are not referred to as components in the chemical engineering literature. Unlike Modelica, the word “component” refers to a particular chemical substance in the process. We call the smallest Modelica components atomic units and the composite Modelica components composite units.

2.5 Notes on the process stream definition

Traditionally, one uses the total molar flowrate, the mole fractions of the chemical substances, the pressure and the temperature to characterize a process stream. In addition, the specific enthalpy is needed to distinguish, for example, between boiling water and saturated steam, as they both have a temperature of 100°C

variable	physical meaning	SI unit
$F \geq 0$	total molar flow rate	[mol/s]
$x[i] \geq 0$	mole fraction of substance $i = 1 : C$	[-]
	$\sum x[i] = 1$	
$p \geq 0$	pressure	[Pa]
$T \geq 0$	temperature	[K]
h	specific enthalpy flowrate	[J/s mol]

Table 2: Traditional choice of variables to characterize a process stream.

at atmospheric pressure. The traditional representation is shown in Table 2.

There are three problems with this representation. (1) The temperature is uniquely determined by the other variables and this relation is nonlinear (equation of state). (2) The material and heat balance equations are nonlinear because mole fractions are used to describe the stream composition. (3) The process stream definition involves an equality constraint (the mole fractions must sum up to 1).

The first two issues make linear atomic unit models nonlinear. In particular, the mixer becomes nonlinear. (The thermodynamically consistent model of the mixer is nonlinear. However, it is practically always made linear in the chemical engineering literature by ignoring the so-called heat of mixing.) The mixer is the only atomic unit having multiple inlets. Thus, a nonlinear mixer has a domino effect: many of the composite units are no longer worth decomposing.

The temperature can be safely dropped from the stream definition. It is uniquely determined by the other variables and it is never needed outside the units. If, for some reason, the temperature of a stream is needed, one can always calculate it by running a flash calculation.

At first sight, it looks strange to the engineer to drop the temperature from the stream definition. Traditionally, the temperature is included in the stream variables (e.g. the EMSO model library, DE P. SOARES & SECCHI [9]) as it is easily measured in real life with a thermometer. Nevertheless, it can be safely excluded.

To make the balance equations linear we use the molar flowrates of the individual substances and the total enthalpy flowrate in place of the total molar flow rate, the mole fractions and the specific enthalpy flowrate. This has the beneficial side-effect that the equality constraint disappears since the mole fractions are not present. With these changes to the stream definition given in Table 2, we arrive at the stream definition presented in Table 1.

2.6 Why not Modelica.Fluid?

The Modelica.Fluid library superficially resembles our library. However, according to the documentation: “*The Modelica.Fluid library provides basic interfaces and components to model 1-dimensional thermo-fluid flow in networks of pipes. [...] there is the restriction that only media models are supported that have T , (p,T) , (p,h) , (T,X) , (p,T,X) or (p,h,X) as independent variables. [...] (Note, T is temperature, p is pressure, d is density, h is specific enthalpy, and X is a mass fraction vector).*”

The Modelica.Fluid library does not aim at supporting chemical process models. Chemical process models are different from flows in networks of pipes.

We want to preserve the linearity of the material and heat balances because it plays an important role in our novel methods. Since the presence of the temperature, the mass / mole fractions or the specific enthalpy would make the balance equations nonlinear, none of them should not appear in the connector class. As already discussed in subsection 2.5, only the *molar* flow rates of the substances, the pressure and the enthalpy flowrate together guarantee linearity. Unfortunately, the Modelica.Fluid library does not allow this choice of the independent variables.

2.7 Hierarchical modeling: composite units

We call the smallest Modelica components atomic units and the composite Modelica components composite units. Often, atomic units only exist on the level of abstraction. For example the equipment in YI & LUYBEN [20] referred to as reactor cannot be decomposed further into smaller, functioning pieces. However, it can be modeled by connecting 7 atomic units and a sink appropriately. None of these units is a reactor. See Figure 3.

The set of atomic units listed in Subsection 2.4 was determined by recursively decomposing a variety of chemical processes. As a result, this set of atomic units is sufficient for general-purpose chemical process modeling.

Figure 4 shows an example of hierarchical decomposition. The vapor-liquid equilibrium cascade is a cascade of stages. A stage is a mixer and a flash unit connected appropriately. In real life, the stages are the smallest, still functioning pieces. The decomposition of the stage into a mixer and a flash unit is an abstraction, as the stage does not have a mixer or a flash unit inside. Nevertheless, this decomposition is valid for modeling.

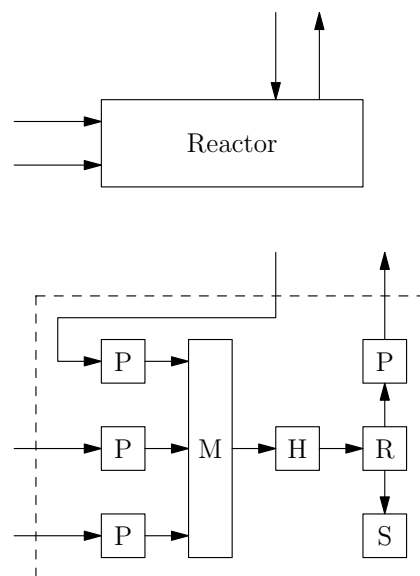


Figure 3: The reactor of Yi & Luyben and its abstract decomposition into atomic units. P: pressure changer, M: mixer, H: heat exchanger, R: reactive flash, S: sink.

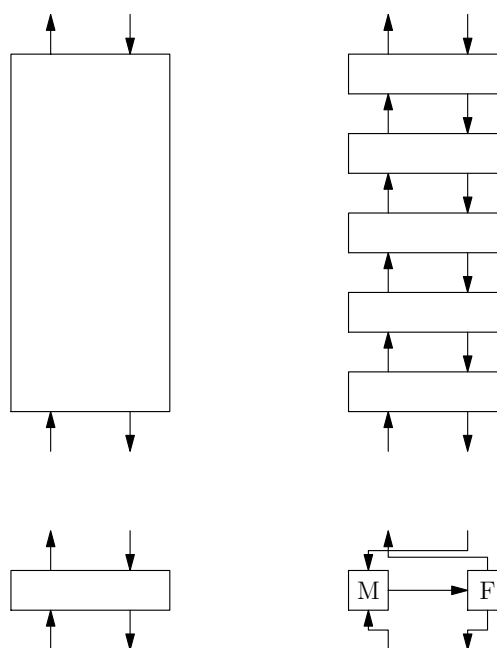


Figure 4: Hierarchical decomposition of the vapor-liquid equilibrium cascade into a cascade of stages, then the decomposition of a stage into a mixer M and a flash unit F.

2.8 Modelica issues encountered

The unit models are valid only if the molar flowrates are nonnegative. This is due to the internal physical structure of the corresponding unit. The natural way to impose these nonnegativity constraints is to impose it on the molar flowrates and the pressure of

the stream, that is, in the connector class. Inequality constraints can be represented within the Modelica language but only by introducing slack variables and setting the min/max on these variables accordingly. This approach is rather inconvenient. The Optimica language extension (ÅKESSON et al. [1]) supports inequalities, it is our preferred way of defining inequality constraints.

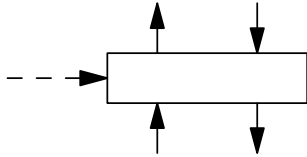


Figure 5: A stage with an optional connection (dashed arrow).

Another difficulty is that Modelica cannot handle arrays of components that have optional connections. All stages have an optional inlet, see Figure 5. This makes the creation of cascades somewhat awkward as missing inlets have to be simulated by dummy streams. The details are difficult to explain in text but easy to understand from the source code. The reader is referred to the source code of the VLEcascade, available from the project homepage at NEUMAIER [17].

3 Application: separation operations

The Modelica implementation discussed in the previous section is tested on a separation operation model. The background of the application is briefly presented. Then numerical results are given for the particular benchmark in subsection 3.2.

A chemical plant takes raw materials as input and produces products as output. Roughly speaking, three steps can be distinguished in a chemical plant: preparation, reaction and purification. See Figure 6. Unwanted chemical substances are separated from the raw input materials in the first step. The unwanted substances may interfere with the reaction in the second step. The reaction produces the desired products and byproducts. Usually a significant fraction of the reactants remain unreacted. These unreacted reactants, the products and the waste byproducts are separated in the third step, called the purification step. The unreacted reactants are recycled, that is, they are fed back to the first step.

Both the first and the third step involves separation operations. In a typical chemical plant, 40–80% of the investment is spent on separation operation equip-

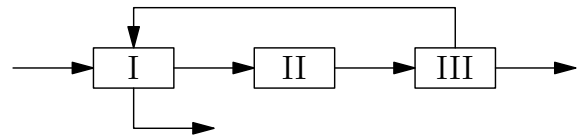


Figure 6: Schematic figure of a chemical plant. Input: raw materials, output: products and byproducts. The steps are (I) preparation, (II) reaction and (III) purification.

ments (PRAUSNITZ et al. [18], p. 2).

Many of the practically relevant equipments used in separation operations (multistage extraction, absorption, desorption, stripping and distillation) are internally a cascade. Not surprisingly, their mathematical model can be solved in a sequential manner.

Identifying multiple steady states is critical to proper design, simulation, control, and operation of these equipments. Unfortunately, professional simulators return only one solution at a time, without indicating the possible existence of other solutions. Usually, only one of the steady-states is desired, the so-called high purity branch. The other steady states are undesirable and potentially harmful as they can lead to unexpected behavior, meaning that the equipment may respond to perturbation in a counterintuitive way.

Given the importance of separation operations, they have already been modeled in Modelica by several authors, for example DURO & MORILLA [11], JOOS et al. [15] and CHANG et al. [7]. Our implementation is based on our Modelica component library for general-purpose chemical process modeling. This distinguishes our implementation from the previous ones.

3.1 Internal physical structure of distillation columns

Distillation columns are used in separation operations. The body of a multistage distillation column is a cascade of stages. In the cascade, the output of one stage is the input of its two neighbors and vice versa, see Figure 4. This structural information can be exploited to solve the underlying process model efficiently.

The internal physical structure is reflected in the mathematical model of the columns. The equations can be evaluated in a sequential manner after guessing just a few variables at one end of the cascade. The essential dimension of the problem is given by the number of variables that have to be guessed to start the stage-by-stage computations. The steady-state model of distillation columns are essentially low-dimensional even if their steady-state model is large-scale.

This approach, reducing the large-scale model to a low-dimensional one, is called the stage-by-stage calculation (LEWIS & MATHESON [16]). Unfortunately, solving the low-dimensional model is very difficult if not impossible with this method, as it shows an extreme sensitivity to the initial estimates. Thus, currently only high-dimensional techniques are in use (DOHERTY et al. [10], 13–33). But a proof-of-concept method remedies the numerical difficulties of the stage-by-stage calculation, see BAHAREV & NEU-MAIER [5].

3.2 Example: multiple steady-states in ideal two-product distillation

The Modelica implementation discussed in Section 2 is tested on the distillation column presented in JACOBSEN & SKOGESTAD [14]. Its main structure corresponds to the linear structure presented in Figure 4, and detailed in subsection 3.1.

Perhaps the simplest distillation columns are the single feed two-product columns with ideal vapor-liquid equilibrium. Even these columns can have multiple steady-states (JACOBSEN & SKOGESTAD [14]). One type of multiplicity can occur when the column has its input specified on a mass or volume basis (e.g., mass reflux and molar boilup). This is of high practical relevance as industrial columns usually have their inputs specified in this way.

The model equations are taken from BAHAREV et al. [4]. Specifications are: methanol-propanol feed composition, mass reflux flow rate and vapor molar flow rate of the boilup. Heat balances are included in the model.

In many studies, one is interested in the dependence of the characteristics on a design parameter (the bifurcation parameter) that can be varied, resulting in bifurcation diagrams. In this case, the design parameter is the reflux flowrate specified on mass basis, and the observed parameter is the product purity. The bifurcation diagram is given in Figure 7. The model equations have five distinct solutions in a certain range of the reflux flow rate. One of the solutions is infeasible in practice because it would result in negative flow rates. The fact that the Modelica implementation gives the expected steady-states suggests that the implementation of the involved atomic and composite units is correct.

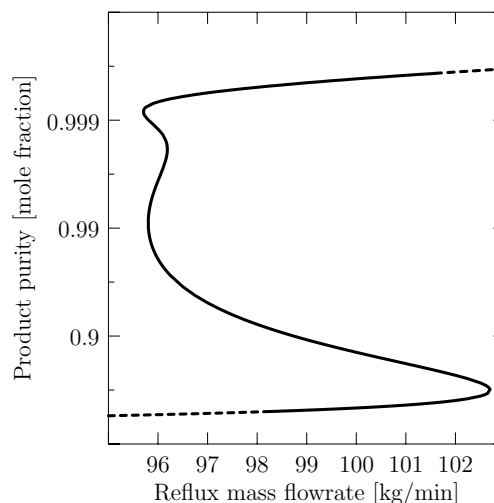


Figure 7: Bifurcation diagram, multiple steady-states in ideal two-product distillation. The infeasible steady-states are represented by dashed lines.

4 Future work

4.1 Recovering structural information

The structural information (connections of the units) can help to solve the underlying process model efficiently, as already mentioned in the introduction and in Subsection 3.1. The core equations of the column in Subsection 3.2 are shown below.

Modelica source:

```
connect(cascade.outVapor,    condenser.inlet);
connect(condenser.distillate,distillate.inlet);
connect(condenser.reflux,   cascade.reflux);
connect(feed.outlet,        cascade.feed);
connect(cascade.boilup,    reboiler.vapor);
connect(cascade.outLiquid, reboiler.inLiquid);
connect(reboiler.bulk,     bulk.inlet);
```

AMPL source:

```
M_eq{j in 1..N-1}:
  sum{k in 1..j} F[k]*z[k] + V[j+1]*y[j+1]
  = D*y[1] + (sum{k in 1..j} F[k]+V[j+1]-D)*x[j];

M_tot:
  F[N_F]*z[N_F] = D*y[1] + (F[N_F]-D)*x[N];

H_eq{j in 1..N-1}:
  sum{k in 1..j} qF[k] + V[j+1]*HV[j+1]
  = V[1]*(HV[1]-HL[0]) + D*HL[0]
  + (sum{k in 1..j} F[k]+V[j+1]-D)*HL[j];
```

The Modelica code is favorable when it comes to structural information, since it speaks about connec-

tions as clearly as possible. The JModelica environment (ÅKESSON et al. [2]) supports programmatic access to the connectivity information. JModelica will play an important role in the further development of our novel methods.

4.2 Optimization

Optimization methods are used in almost all areas of engineering. Typical problems in chemical engineering arise in process design, process control, model development, process identification and real-time optimization, see BIEGLER [6]. Our ultimate goal is to develop structure-driven optimization methods for solving nonlinear programming problems (NLP). This requires an objective function (e.g. minimize cost, maximize yield or profit) to be included in the model. Inequality constraints often required too. Unfortunately, Modelica does not support cost function and inequalities, only the Optimica language extension (ÅKESSON et al. [1]) does.

4.3 Dynamic simulation

At the moment, only the steady-state model equations of the units are implemented in Modelica. It is possible to extend the library to support dynamic simulation, but it is not easy in practice. Often, the model equations are not accurately known and the dynamic calculations may involve additional pitfalls.

Acknowledgements. The research was funded by the Austrian Science Fund (FWF): P23554.

References

- [1] Johan Åkesson, Tove Bergdahl, Magnus Gäfvert, and Hubertus Tummescheit. Modeling and Optimization with Modelica and Optimica Using the JModelica.org Open Source Platform. In *Proceedings of the 7th International Modelica Conference, Como, Italy, 20-22 September 2009*, pp. 29–38. Linköping University Electronic Press, Linköpings universitet, 2009.
- [2] Johan Åkesson, Torbjörn Ekman, and Görel Hedin. Implementation of a Modelica compiler using JastAdd attribute grammars. *Science of Computer Programming*, 75:21–38, 2010.
- [3] Johan Åkesson. *Languages and Tools for Optimization of Large-Scale Systems*. PhD thesis, Regler, nov 2007.
- [4] Ali Baharev, Lubomir Kolev, and Endre Rév. Computing multiple steady states in homogeneous azeotropic and ideal two-product distillation. *AIChE Journal*, 57:1485–1495, 2011.
- [5] Ali Baharev and Arnold Neumaier. Steady-state multiplicities in reactive distillation: stage-by-stage calculation revisited. *AIChE J.*, 2012.
- [6] Lorenz T. Biegler. *Nonlinear programming: concepts, algorithms, and applications to chemical processes*. SIAM, 2010.
- [7] Chen Chang, Ding Jianwan, Chen Liping, and Wu Yizhong. Media Modeling for Distillation Process with Modelica/Mworks. In *Proceedings of the 8th International Modelica Conference, March 20th-22nd, Technical Univeristy, Dresden, Germany*, pp. 239–245. Linköping University Electronic Press, Linköpings universitet, 2011.
- [8] Amy R. Ciric and Peizhi Miao. Steady state multiplicities in an ethylene glycol reactive distillation column. *Ind. Eng. Chem. Res.*, 33:2738–2748, 1994.
- [9] R. de P. Soares and A. R. Secchi. EMSO: A new environment for modelling, simulation and optimisation. In *Computer Aided Chemical Engineering*, vol. 14, pp. 947–952. Elsevier, 2003.
- [10] M. F. Doherty, Z. T. Fidkowski, M. F. Malone, and R. Taylor. *Perry's Chemical Engineers' Handbook*. McGraw-Hill Professional, 8th ed., 2007.
- [11] N. Duro and F. Morilla. A Modelling Methodology for Distillation Columns using Dymola and Simulink. In *Applied Simulation and Modelling*. ACTA Press, 2003.
- [12] Robert Fourer, David M. Gay, and Brian Wilson Kernighan. *AMPL: A Modeling Language for Mathematical Programming*. Brooks/Cole USA, 2003.
- [13] Peter Fritzson. *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*. Wiley-IEEE Press, 2004.
- [14] E.W. Jacobsen and S. Skogestad. Multiple steady states in ideal two-product distillation. *AIChE Journal*, 37:499–511, 1991.

- [15] Andreas Joos, Karin Dietl, and Gerhard Schmitz. Thermal separation: An approach for a modelica library for absorption, adsorption and rectification. In *Proceedings of the 7th International Modelica Conference, Como, Italy, 20-22 September 2009*, pp. 804–813. Linköping University Electronic Press, Linköpings universitet, 2009.
- [16] W. K. Lewis and G. L. Matheson. Studies in distillation. *Industrial and Engineering Chemistry*, 24:494–498, 1932.
- [17] Arnold Neumaier. Structure-driven methods for large-scale optimization, 2012. URL <http://www.mat.univie.ac.at/~neum/structure.html>.
- [18] John M. Prausnitz, Rüdiger N. Lichtenthaler, and Edmundo Gomes de Azevedo. *Molecular Thermodynamics of Fluid-Phase Equilibria*. Prentice Hall PTR, Upper Saddle River, NJ, third ed., 1999.
- [19] Carl Sandrock and Philip L. de Vaal. Dynamic simulation of chemical engineering systems using OpenModelica and CAPE-OPEN. In *19th European Symposium on Computer Aided Process Engineering*, vol. 26 of *Computer Aided Chemical Engineering*, pp. 859–864. Elsevier, 2009.
- [20] Chang K. Yi and William L. Luyben. Design and control of coupled reactor/column systems—Part 1. A binary coupled reactor/rectifier system. *Computers & Chemical Engineering*, 21(1):25–46, 1996.

FMI Add-on for NI VeriStand for HiL Simulation

Cosimo Palma Marco Romanoni
Dofware S.r.l.

10099 San Mauro Torinese (Torino, Italy)

cosimo.palma@dofware.com marco.romanoni@dofware.com

Abstract

Currently, most of the links from Modelica models to real-time hardware platforms suitable for Testing and Validation are based upon non standard model exchange format, or rely on third party preprocessor.

This paper describes the implementation of the Modelisar Functional Mock-up Interface (FMI) support in NI VeriStand, a commercial software environment suitable for real-time testing applications.

This paper presents the work conducted to implement the FMI Add-on for NI-VeriStand, which is available as a commercial product, and the process to make hardware in the loop simulation starting from a Model Based Development environment compliant with the FMI for Co-Simulation standard for model export and using it in NI VeriStand environment with National Instruments real-time hardware.

Keywords: FMI; Hardware in the Loop; NI VeriStand; Real Time Systems

1 Introduction

FMI stands for “Functional Mock-up Interface” [1] and is an open standard for model exchange specified in the ITEA2 Modelisar project [2]. The aim of this work is to enable NI VeriStand [3] to support the FMI standard for Co-Simulation. This in order to perform rapid-prototyping and hardware in the loop simulations using National Instruments hardware directly from Modelica models exported using the FMU standard. With the FMI Add-on it is possible to use FMU models in Windows and /or in National Instruments RT Targets like NI PXI [4] and NI CompactRIO [5].

In this paper, we will present:

- A description of the activity carried out for the implementation of the FMI Add-on [6] for NI-VeriStand.

- A detailed description of the steps that are to be performed in order to use FMUs in National Instruments PXI RT Targets.
- A validation test for the FMI Add-on performed with Dymola [7] and National Instruments PXI RT Target based on the detailed model of a 6 dof manipulator.

2 Scenario

Using the FMI Add-on you can make MiL/SiL/HiL with NI VeriStand framework and all Model Based Design environments compliant with the FMI for Co-Simulation standard. Figure 1 shows a typical scenario of the automotive field where some control algorithms have been designed in Simulink and LabView, and the car model has been modeled in Dymola. With the FMI add-on and NI VeriStand it is possible to deploy the plant model along with the control algorithms in several targets and perform HiL Validation for the whole system.

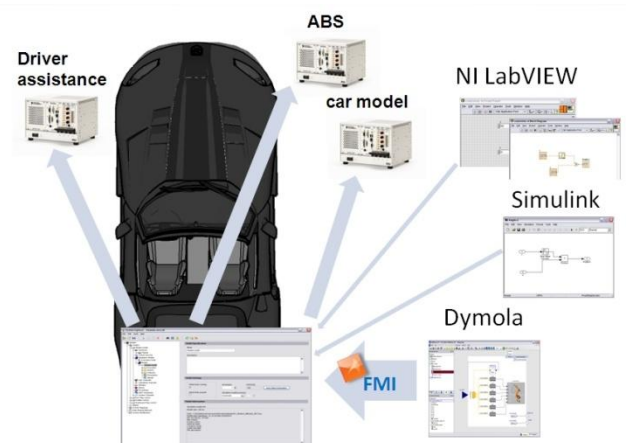


Figure 1: Example of HiL scenario in the automotive field

3 NI VeriStand

NI VeriStand is a software environment for configuring real-time testing applications. NI VeriStand helps you configure a multicore-ready real-time engine to execute tasks that include the following:

- Real-time stimulus generation
- Analog, digital, and communication bus interfaces
- Real-time stimulus generation
- Analog, digital, and communication bus interfaces
- Field-programmable gate array (FPGA)-based I/O interfaces
- Calculated channels
- Triggerable, multfile data logging
- Event alarming and alarm response routines

NI VeriStand can also import control algorithms, simulation models, and other tasks from NI LabVIEW software and third-party environments. You can monitor and interact with these tasks using a runtime editable user interface that includes many useful tools for value forcing, alarm monitoring, I/O calibration, and stimulus profile editing.

4 Co-Simulation

Co-Simulation is an approach for joint simulation of models developed with different tools where each tool treats one part of a modular coupled problem. Intermediate results are exchanged between these tools during simulation where data exchange is restricted to discrete communication points.

Between these communication points the subsystems are solved independently. Figure 2 shows an example of Co-Simulation where a complete system has been modeled using three different tools, and where each model uses its own numerical solvers and exchanges data thanks to the Co-Simulation master environment during the simulation.

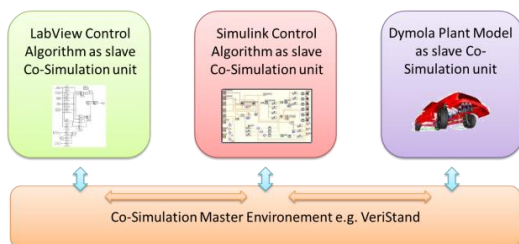


Figure 2: Example of Co-Simulation scenario in the automotive field with Dymola, Simulink and LabView

5 Implementation

5.1 FMU and Real Time Target

Most Modeling and Simulation Environments compliant with the FMI 1.0 specification export an FMU file that contains an XML file and a Dynamic link library (dll) in order to maintain the intellectual properties of the model and the integrator algorithms.

For this reason the first issue that arose in the development of the FMI add-on was how to use the model compiled as dll on RT Target running an operating system different from Windows.

Most of National Instruments Real Time Targets use Phar Lap ETS as operating system. Phar Lap ETS is a dedicated real-time operating compliant with a subset of Windows Application Programming Interface (win32 API) [8].

As consequence the dll included in the FMUs generated by the commercial modeling tools had to be checked against Phar Lap requirements.

The first tool chosen was Dymola from Dassault Systèmes, whose FMU generation routines were modified by Dassault Systèmes development team in order to solve all compatibility issues.

In order to check the compatibility of the dll included into the FMU files, LabVIEW RT DLL Checker [10] has been used. Using this tool, dll generated by any commercial or free tool, together with hand coded and compiled ones must be tested in order to check if they are compliant for Phar Lap ETS environments.

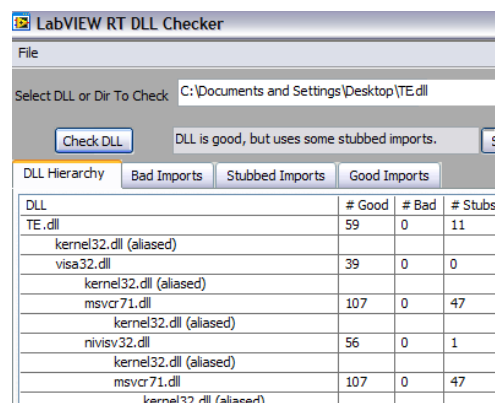


Figure 3: LabVIEW RT DLL Checker window

If an FMU model passes this test, it is suitable for its use into RT Phar Lap Operative Systems such as National Instruments PXI platforms.

The implementation that is been performed to use FMUs into NI VeriStand consists in a wrapper between the FMU specifications and NI VeriStand approach to process simulation models.

We used C/C++ code for the part that has to be deployed on RT target and C# language for interact with NI VeriStand interface for acquire user specification for the model and the simulation.

5.2 Co-Simulation Master and Slave

NI VeriStand environment is a full featured Co-Simulation platform working as master for the Co-Simulation process. When several FMU's and Co-Simulation slave models exported using other model exchange formats, e.g. S-Functions, are imported into NI VeriStand, it works as master considering every imported model as slave.

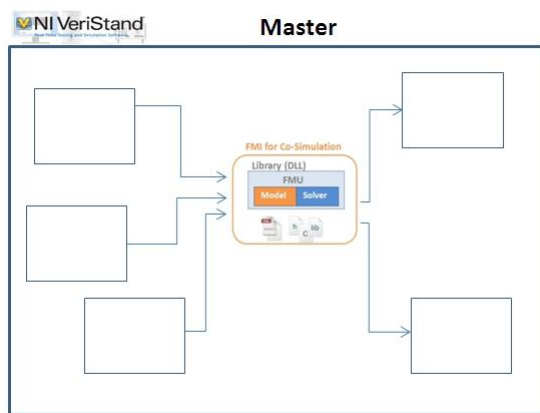


Figure 4: NI VeriStand logical schema during Co-Simulation

Data exchange between subsystems is restricted to discrete communication points. In the time between two communication points subsystems are solved independently from each other by their individual solver. NI VeriStand controls the data exchange between subsystems and the time synchronization of all slave simulation solvers.

A simulation model can be coupled if it is able to communicate data during simulation at certain time points.

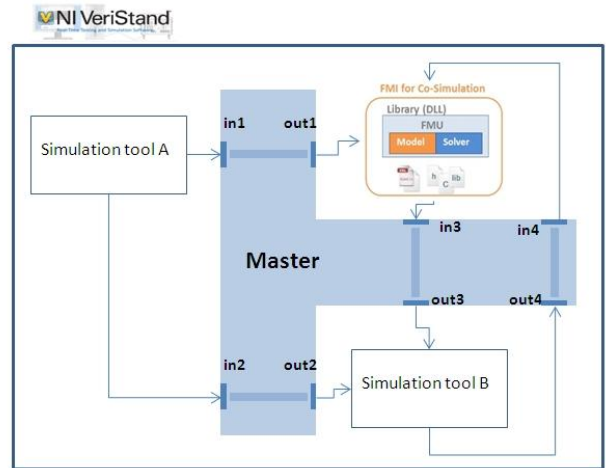


Figure 5: NI VeriStand logical schema during Co-Simulation

In NI VeriStand you can define the duration of the time step in which VeriStand will read the inputs and write the outputs. A time step is the atomic unit of time that all simulation tasks needed to be completed.

6 Using FMU models on NI Real Time

This section describes the process of validation for a model exported as FMU using an RT hardware platform.

First of all a general schema of the HIL scenario, depending on the system that needs to be validated has to be developed, see Figure 1. Once the HIL schema has been decided, the installation of the FMI Add-on must be done in each RT target that will host an FMU model, see Fig. 6.

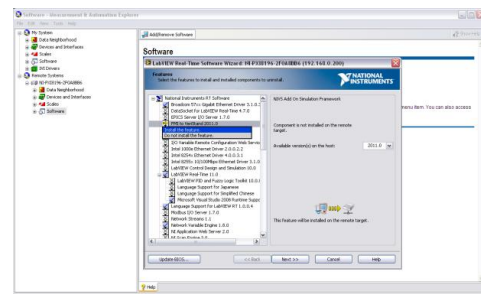


Figure 6: Installation of the FMI add-on on NI RT targets

In order to setup the HIL system platform, each subsystem model must be exported from the model

based environment in which it was developed. With the installation of the FMI add-on VeriStand will support fmu files in addition to dll, lvmodel, mdl and out files (Fig. 8) . It has to be highlighted that FMU models must be compliant with the FMI for Co-Simulation 1.0 standard. Fig. 7 shows a Dymola model exported with the FMI standard.

1. Export Model as FMU

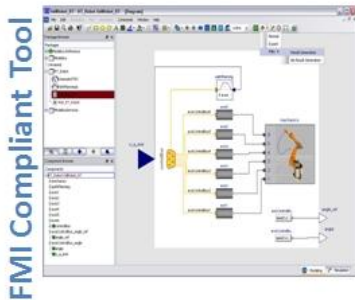


Figure 7: FMU for Co-Simulation model export from Dymola

After each sub-model has been exported, it must be imported into NI VeriStand, using the NI System Explorer as shown in Fig. 8.

2. Import FMU Model

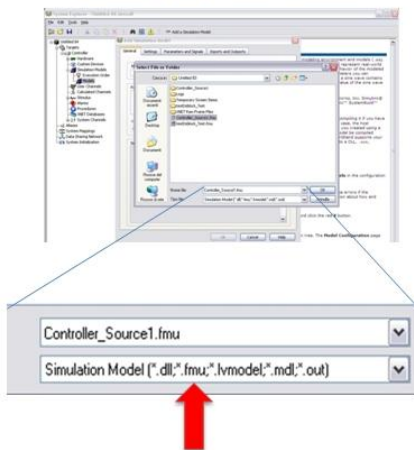


Figure 8: FMU model import using the NI System Explorer

Once the models have been imported into VeriStand, connections must be configured in order to map hardware acquisition boards I/O with model variables.

After System Explorer configuration a suitable workspace can be configured into NI VeriStand in order to control inputs and visualize the outputs in real-time, see Fig. 9. Finally the deployment on the targets will be performed automatically by VeriStand and users can use their system models in HiL.

3. Deploy and Validate



Figure 9: NI VeriStand workspace with custom scopes

7 Test and validation of the FMI add-on

In order to test the performance and validate the results of the HiL simulations performed on RT targets using FMU models, a detailed model of a six dof mechanical manipulator has been chosen as a test case. The model of the physical system is a version of the *Modelica.Mechanics.MultiBody.Examples.Systems.RobotR3.fullRobot* that can be found in the Modelica Standard Library, modified using Dymola, see Fig. 7. The original model has been modified adding one real input for each degree of freedom of the manipulator as reference angle for the motion of the electric drives. The new model can be controlled by external sources in real-time. We used also two simple control algorithms developed, one in Simulink and the other one in LabView, to control two of the inputs of the System. The purpose of this two control algorithms was to demonstrate the capabilities of this methodology and the scalability of the system architecture.

7.1 The system architecture

The architecture used to validate the system is shown in Fig. 10, and consists in three models exported in different formats from different tools that will be executed together exchanging data in real-time on a NI PXI-8196 Phasor Lap RT target.

The target has been configured into VeriStand using the System Explorer, as shown in Fig. 11. Two different validation campaigns have been performed, the first using Windows and the second using Phasor Lap on NI PXI-8196 as targets.

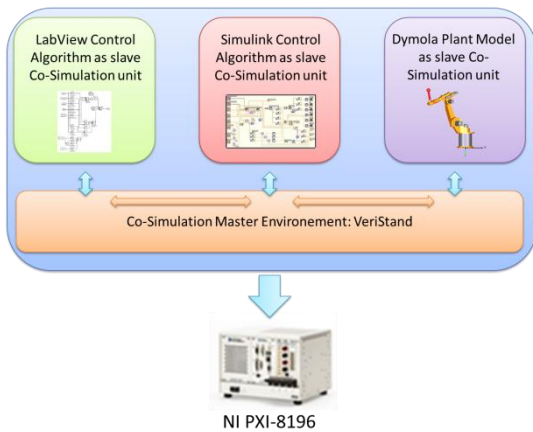


Figure 10: System architecture for the Validation experiment

Thanks to the Dofware FMI add-on, the FMU model was loaded into VeriStand and the *depvs* file, needed for the deployment of the project into the target, automatically generated.

Thanks to the support of National Instruments development team it was possible to grant the best user experience in VeriStand to FMU importers. In fact the FMI add-on was developed in a way so that the import process in VeriStand is now the same for every model exchange format, as shown in Fig 11.

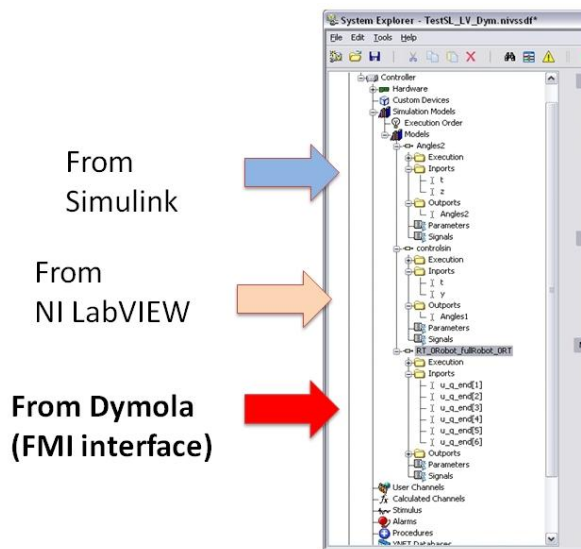


Figure 11: Import of different external models using the system explorer

It is important to note that model parameters imported using the FMU format must be set in the modeling environment before the export phase. This is because the FMI specification does not allow parameter tuning during the simulation phase but only before the initialization phase of the model and this cannot be done into VeriStand. For this reason all the parameters of the model that need to be tunable dur-

ing the HIL validation tests have to be modified and changed into inputs before the FMU generation. Still the parameters of the imported models can be seen in the system explorer as shown in Fig.12.

This issue will be more likely solved with the next release of the FMI specification that will allow parameter tuning during runtime simulations.

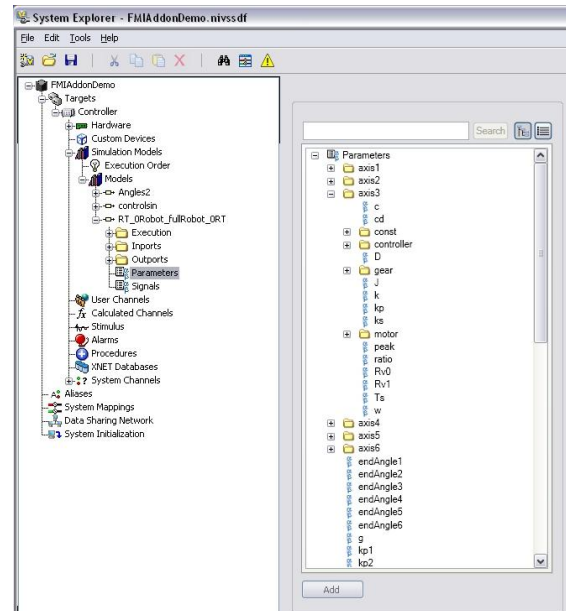


Figure 12: Model parameters explorer into VeriStand

To finish the setup phase of the system architecture, the I/O of the all models must be coupled together using the System Configuration Mappings editor in VeriStand, see Fig. 13, and with physical I/O of the target, see Fig. 14.

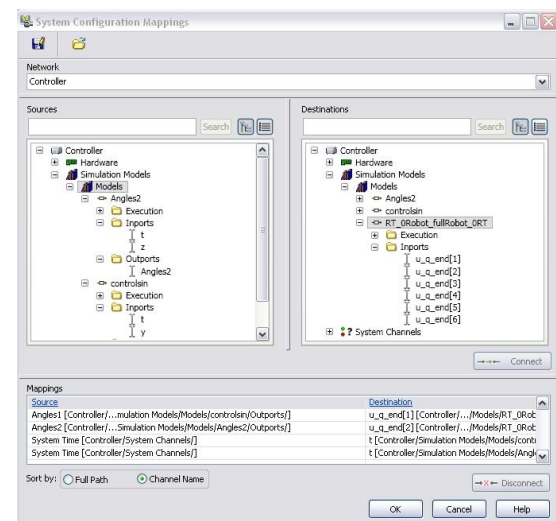


Figure 13: System Configuration Mappings editor in VeriStand, I/O of the models are coupled together

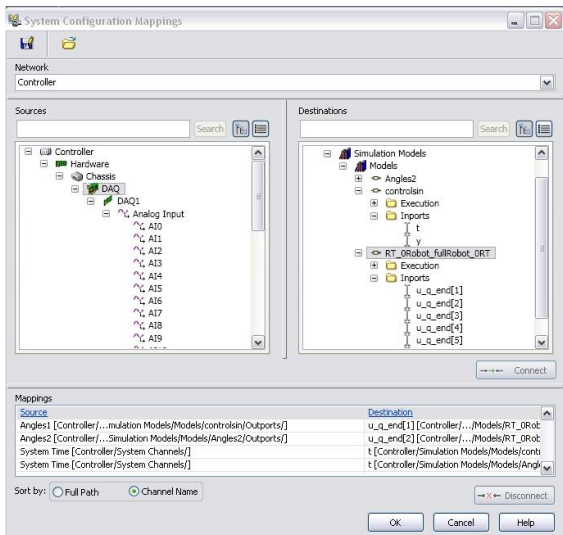


Figure 14: System Configuration Mappings editor in VeriStand, I/O of the models are coupled with physical I/O of the DAQ systems

VeriStand will drive the Simulation, thus the time step along with the execution order of the real time task models have to be set into the NI tool.

It is important to note that the solver’s time steps of the single slave models must be smaller than the sampling time of the master device. It is also important to note that the execution time of a single time step of each single slave model must be smaller than the allocated running time in VeriStand, or the simulation will be shut down.

The allotted running time is determined by the interaction of your models and the hardware in your system. In fact VeriStand must complete all tasks in addition to the ticking of the models, such as input data processing and output data returning. The number of inputs and outputs in the system might increase the time that has to be allotted for a time step. Moreover if the system includes multiple models, all of your models might need to perform a task during each given time step.

7.2 I/O Monitoring and validation results

At this point we are able to deploy the project into the target, i.e. all FMU resource files are copied to the target accordingly to the *depvs* files and the simulation can begin.

Once the simulation is running, the simplest way to monitor the state of the outputs is to develop a custom workspace into VeriStand, see Fig. 15. The workspace can include scopes for the model variables and interactive controllers for input and parameter real-time tuning.



Figure 15: NI VeriStand workspace developed to control the manipulator model

Fig. 16, 17, 18 and 19 show the simulation results in terms of reference angles and simulated angles for each axis of the manipulator. The results are compared to the ones obtained simulating the same system entirely in Dymola. We can note that the simulation results in Dymola are matching with the ones obtained in VeriStand using both windows and NI PXI targets. It has also to be noted that there are some little differences from the results obtained in Dymola and the ones obtained in the PXI target generated from the “errors” introduced by different solvers and by logger used to save the output results and the limited resources of the PXI that generated data losses in the communication between the sub-model units and the master sample time. The data logger was added as a custom device in VeriStand, and is capable of saving the output data on the file system using a fifo.

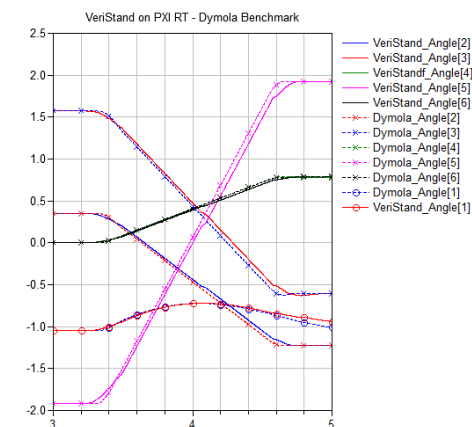


Figure 16: NI VeriStand with PXI target Vs Dymola benchmark

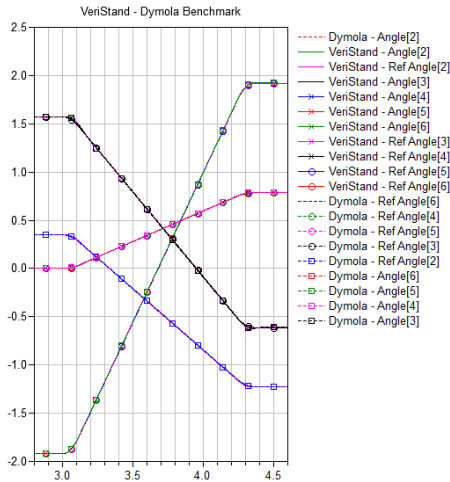


Figure 17: NI VeriStand with Windows target Vs Dymola benchmark

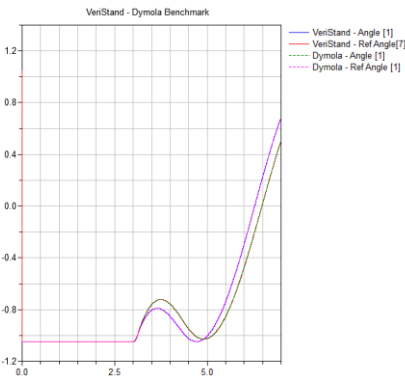


Figure 18: NI VeriStand with Windows target Vs Dymola benchmark

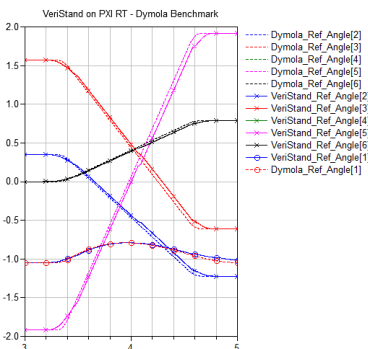


Figure 19: NI VeriStand with PXI target Vs Dymola benchmark

8 Conclusion

Thanks to the development of the FMI add-on for NI VeriStand a tool chain can now be defined to perform HIL simulation for control validation on NI hardware platforms starting from Modelica based models.

The tool chain has been validated against simulation results obtained in Dymola on a detailed system model divided into sub-systems in order to validate also the data communications between sub-models during the HIL tests. NI VeriStand present the possibility to use an external solver, so we created an add-on also for FMI for model-exchange. We successfully tested it also on Phar Lap OS, but so far very few solvers have been implemented (Euler and Runge-Kutta). This solution gave us good results for simple models, but has still to be improved in order to implement solvers capable of efficiently handle complex models.

An important milestone on the roadmap of the FMI add-on will be the compatibility upgrade with respect to the FMI 2.0 specification. In this way no more modifications will be needed in order to tune the parameters of the Modelica models during HIL validations.

References

- [1] Functional Mock-up Interface: <http://www.functional-mockup-interface.org/index.html>
- [2] MODELISAR consortium. Functional Mock-up Interface for Model Co-simulation V. 1.0. <http://www.modelisar.org>
- [3] NI VeriStand: <http://www.ni.com/veristand/>
- [4] NI PXI: <http://www.ni.com/pxi/>
- [5] NICompactRIO: <http://www.ni.com/compactrio/>
- [6] FMI Add-on for NI VeriStand, <http://www.dofware.com/products/fmi-add-on-for-ni-veristand/>
- [7] Dymola <http://www.3ds.com/products/catia/portfolio/dymola>
- [8] FMI for Co-Simulation http://www.modelisar.com/specifications/FMI_for_CoSimulation_v1.0.pdf
- [9] FMI for Model Exchange and Co-Simulation, version 2.0 <http://www.modelisar.com/specifications/FMI>

[I for ModelExchange and CoSimulation v 2.0 Beta3.pdf](#)

- [10] LabVIEW RT DLL Checker
<http://digital.ni.com/public.nsf/allkb/0BF52E6FAC0BF9C286256EDB00015230>
- [11] QTronic GmbH: FMU SDK 1.0.1
<http://www.qtronic.de/en/fmusdk.html>
- [12] Modelica Association: Modelica – A Unified Object-Oriented Language for Physical Systems Modeling. Language Specification, Version 3.2. March 24, 2010. Download: <https://www.modelica.org/documents/ModelicaSpec32.pdf>
- [13] Elmqvist H., Otter M., Henriksson D., Thiele B., and Mattsson S. E. Modelica for Embedded Systems. In Proceedings of the 7th Modelica Conference, Como, Italy, September, 2009.
- [14] H. Hadj-Amor, C Faure, M. Ben Gaïd, N. Pernet, “Towards a Modelica Real-time co-simulation with FMI”, Multiphysics Simulation - Advanced Methods for Industrial Engineering Conference, Fraunhofer, 22-23 June 2010.

Using Static Parametric Design to Support Systems Engineering of Industrial Automation Systems

Hongchao Ji¹ Lars Mikelsons¹ Karl Kempf¹ Dieter Schramm²

¹Bosch Rexroth AG, Lohr am Main, Germany

{hongchao.ji, lars.mikelsons, karl.kempf}@boschrexroth.de

²University of Duisburg-Essen, Duisburg, Germany
dieter.schramm@uni-due.de

Abstract

This paper proposes a static parametric design methodology for application of the model based systems engineering (MBSE) paradigm in the world of Modelica. This methodology allows for parameter synthesis of the industrial automation systems under consideration of customer requirements. Furthermore, the parametrized system can be verified automatically. An integrated system model consisting of requirements, system design and verification models is created and can be used as a design template to generate a new parameter set according to the change of customer requirements. A case study from the practice is presented to proof the concept of this methodology.

Keywords: Model Based Systems Engineering, SysML, Modelica, Parameter Synthesis

1 Introduction

The complexity of modern industrial automation systems increases steadily. New functions and technologies need to be integrated to fulfill customer requirements, environmental regulations and/or safety standards. The increasing complexity has raised many challenges such as keeping the the design consistent and approving the correctness with respect to the customer requirements. Model based systems engineering (MBSE) is defined as the formalized application of modeling to support system requirements, design, analysis, verification and validation activities beginning in the conceptual design phase and continuing throughout development and later life cycle phases. Hence MBSE is a suitable approach to cope with these challenges.

One of the key issue in MBSE process is to determine the proper dimension of the system design ac-

ording to the formalized requirements model. The static parametric design methodology uses Modelica static models together with the dynamic models to support the MBSE process by the means of selecting the proper components of the desired system from given product catalogs, dimensioning the sub-systems as well as checking the correctness of the system design with respect to systems requirements.

The objective of the static parametric design methodology is to perform a parameter synthesis of a technical system according to the customer requirements automatically. Furthermore, the calculated system design can be verified automatically as well. The Systems Modelling Language (SysML) [11] is used to formalize the customer requirements. Moreover, the extension of abstraction levels and classification defined in [4] is also applied in this paper. Work on the integration of SysML and Modelica has already proven its effectiveness in the MBSE [6, 8, 9]. Reusing these improvements the SysML models can be transformed into executable Modelica models.

In this contribution, we focus on the standard application that the structure of the desired system is normally known to the system engineers. The requirements models serve as the basis of the static parametric design methodology. By changing the requirements models, a new parameter set of the system can be obtained automatically. In this sense, the integrated model consisting of requirements, system design and verification models can be seen as a design template for a standard application.

This paper is organized in 6 sections. Section 2 illustrates the current systems engineering process and the need of model based systems engineering and static parametric design methodology. As part of related work in Section 3 a short introduction to SysML and its integration with Modelica are given. Section 4 introduces the static parametric design methodology

to support the systems engineering in detail. The capabilities of the proposed methodology are demonstrated using an industrial application in Section 5. The paper closes with conclusions and an outlook to future work.

2 Systems Engineering of Industrial Automation Systems

The systems engineering process is described in the following referring to the well known V Model according to the VDI 2206 standard [12] depicted in (Figure 1).

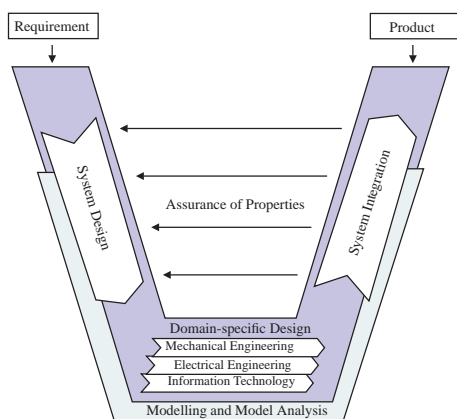


Figure 1: V Model According to VDI 2206 [12]

The main tasks of current systems engineering of industrial automation systems can be summarized as follows:

- State the customer needs correctly and unambiguously;
- Define the proper system design based on the customer requirements;
- Verify the system design against customer requirements.

They will be introduced respectively in the following.

2.1 Requirements Specification

Due to the fact that the requirement specification is the subject matter of contract between customer and contractor the above described context implies that the requirements engineering has to be seen not only from the technical perspective but also needs to consider the business process and the contractual situation along the supplier chain. In this context it is self-evident that the requirements shall be defined and structured not only according to technical aspects but also according

to the contractual situation. The definition of levels of abstraction is an appropriate way to meet these needs. The depicted levels of abstraction in Figure 2 reflect the described supplier chain and major technologies involved and therefore are a reasonable choice in the context of automation systems. In order to deal with the complexity of large systems the design objects are clustered in a system break down structure. The requirements derived on the different levels of abstraction can be referenced in requirement specifications in order to provide the contractual views on subsets of requirements.

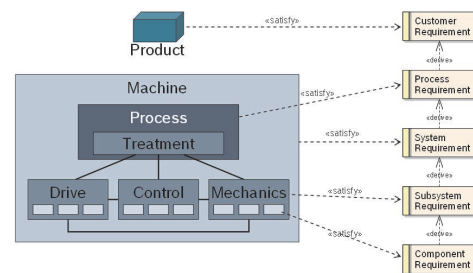


Figure 2: Levels of Abstraction in Requirements and System Design

2.2 Systems Design

Industrial automation systems are characterized by their ability to process a material or work piece according to a defined procedure to achieve the output of a desired product. The challenge of the system engineer is to design a machine that is capable to run the process in a deterministic and efficient way. This task is typically performed within a specific design domain that refers to a field of technical expertise. The proper selection of the components and their integration into the overall structure strongly influence the function, performance, robustness and reliability of the whole system. Currently the selection of the components is mainly determined by the competence of the system engineers which is time consuming and error prone. In order to avoid manual errors, it is desired to select the components in a systematical manner.

Today, parameter synthesis of a technical system is usually based on static calculations in the field of industrial automation systems. A small example of such static calculation for selection of a hydraulic valve in a hydraulic lift system is depicted in Figure 3. The dimension of the cylinder shall be first defined in order to calculate the maximal flow rate through the valve

with the given maximal cylinder velocity by

$$Q_{max} = A_D \cdot v_{max}. \quad (1)$$

After that the nominal size of the valve is determined by some design criteria. Moreover, the design criteria origin from customer requirements as well. In this case, the nominal size is defined by the nominal flow rate which is calculated by

$$Q_{nominal} = 1.5 \cdot Q_{max}. \quad (2)$$

The nominal size of the valve can be chosen from the product catalog according to the nominal flow rate. That means a proper component, in this case the valve, is correctly selected for the desired system.

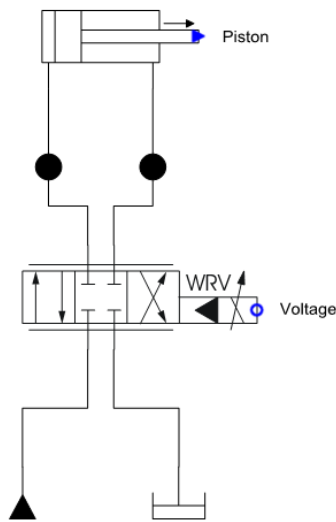


Figure 3: Schematic of a Hydraulic Lift System

In order to ensure the acceptance of a MBSE tool offering the methodology presented in this paper, these design guidelines have to be integrated into the MBSE tool.

2.3 Verification and Validation

Model based verification and validation of a systems design against systems requirements has been widely used in the field of industrial automation systems. The goal here is to verify the system design in an automated and reproducible way. Since this issue has already been addressed in virtual verification of systems design against systems requirements (vVDR) methodology [10], it is used to approve the systems design in this paper as well.

2.4 The Challenges

Since industrial automation systems usually consist of components from different domains, it is hard to keep the design correct and consistent. In order to deal with this fact, the systems design from different engineering domains shall be integrated into the whole MBSE process. Therefore, a universal and standardized modeling language is required which shares the understanding among engineers from different disciplines. This common language shall enable the generation of requirements models, system design models, traceability models as well as verification models containing domain-specific details. SysML is being proposed to meet this requirement. However it has been evaluated as not sufficient due to the lack of executable semantics. Integration of the languages SysML and Modelica has proven its efficiency in the area of MBSE [5, 6, 7, 10]. Therefore, in this contribution SysML and Modelica are chosen as the modelling languages applied in our MBSE process as well.

In order to set up a MBSE tool for parameter synthesis, the following two questions have to be answered:

1. How to perform a parameter synthesis to determine all the proper components based on the customer requirements in an automatic manner?
2. How to link different kinds of models in the whole MBSE process?

These two challenges have been addressed in this paper by the means of using static parametric design methodology in an integrated system model based on SysML and Modelica. The details are presented in Section 4.

3 Background and Related Work

3.1 The SysML and Modelica

SysML is a general purpose language used in the field of systems engineering. It is defined as a UML profile which reuses subsets of UML constructs and extends them with some additional modeling elements. SysML is capable to capture the textual requirements and to allocate them with the design models and test cases. However due to loosely defined executable semantics SysML is not capable to execute the modeled physical systems. In contrast to that, Modelica is an object oriented and equation based modeling language for multi-domain physical systems. Graphical modeling is supported by the object diagram which offers an

intuitive way to describe power transmission through acausal connections as well as directed signal flows. Strong semantics allow the generation of executable models of continuous as well as discrete systems. Object oriented language constructs enable the efficient reuse of models and the design of comprehensive and easy to use model libraries. As mentioned in Section 2, a language which integrates the descriptive modeling power of SysML and the formal executable simulation power of Modelica seems to be a promising approach for the systems engineering in industrial automation systems.

3.2 Related Work

Several work has already been done towards application of MBSE paradigm using Modelica language with different concerns. The vVDR methodology [10] addresses mainly the virtual verification of systems requirements by using UML, Modelica and ModelicaML. In Dubois et. al. [2] a requirement traceability model to enforce the traceability concept in SysML in the automotive domain is presented. Requirements management and allocation have already been covered in the other paper of the author [4].

This paper describes a methodology for the parameter synthesis of technical systems according to customer requirements. Moreover, the different kinds of models are linked with the other models in the integrated system model and therefore it is easy to regenerate and to verify the final parametrized system. They will be introduced in detail in the following section.

4 Static Parametric Design

The proposed static parametric design methodology is based on Modelica static models. Static models are defined as models that are constant over time. Considering the whole MBSE process, the following items can be formalized as Modelica static models:

- *Requirement Specifications*,
- *Product Catalogs*,
- *Selection Criteria*.

4.1 Definitions of Models

The requirement specifications can be defined as *requirements models* which are captured as stereotyped SysML models according to the different abstraction levels and classifications in [4].

This classification is mainly based on the taxonomy proposed in [3] with some changes as presented in the following. Instead of the requirement type specific quality, the structural requirement is defined in the field of industrial automation.

- A *functional requirement* is the requirement that should produce an expected reaction to a given stimuli.
- A *performance* is the requirement to check whether a system variable such as timing, speed, volume or throughput is in a desired range.
- A *structural requirement* is the requirement which describes the structural demand of the stakeholder.
- A *constraint* is the requirement to provide the technical and safety boundary conditions that the system shall satisfy.

The *product catalogs models* are easy to understand as Modelica static models. They can be modeled as a simple record class with some table definitions. In this work, a UML library with a sub-set of the Bosch Rexroth catalog is implemented and later transformed to Modelica static models. The advantages of implementation as UML library over Modelica library is the compatibility with SysML and the extendibility of the product catalog.

The selection criteria are implemented as *static calculation models*. The idea is using Modelica functions to determine the proper dimension of the components. Normally, those selection criteria for the components are usually the same. Due to the fact of reuseability, a Modelica library called *ParametricDesign* which consists of most selection criteria in the field of industrial automation systems is built as shown in Figure 4.

Besides the static models, *simulation model* is an executable model which is used for dynamic simulation. In the verification and validation phase, the simulation model is linked with the test cases to check the correctness of the parametrized system design.

All those models are further transformed into Modelica static models with the help of a Modelica code generator, which is implemented with the help of Eclipse Aceleo [1].

4.2 Link of Different Models

Several models have been defined in this static parametric design methodology. It is necessary to link

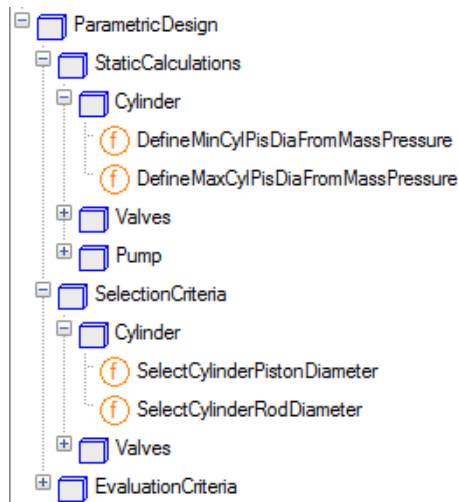


Figure 4: Structure of the Parametric Design Library

those models in a efficient manner in order to perform a parameter synthesis automatically. The basic idea is to reference the attributes of SysML model to the variables, parameters and constants of the Mod- elica model. Currently, these relations have been es- tablished manually which is time consuming and error prone. A method to extend the standard relationships such as «satisfy», «verify» and «derive» has been pro- posed in [4]. An overview of the linking of different models in this methodology is shown in Figure 5. The tooling that supports the binding of related objects is implemented in Eclipse.

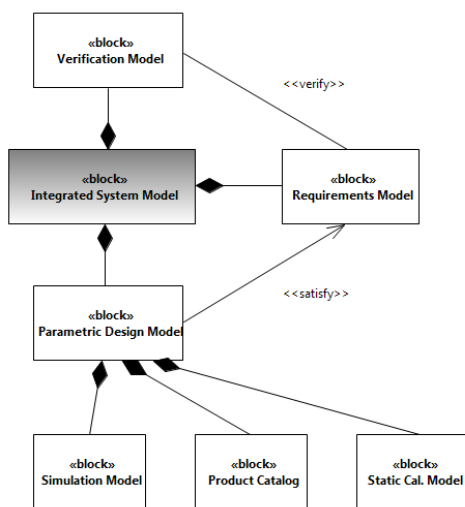


Figure 5: Link of Different Models

4.3 Methodology Description

The prerequisites of application of the static design methodology are

- *hydraulic library,*
- *static design library,*
- *product catalog library.*

Furthermore, the formalized requirements models and at least one simulation model have to be created at first as the basis for application of the static parametric de- sign methodology.

The main steps of this methodology can be summa- rized as follows:

1. Capture the customer requirements as stereo- typed requirements according to the proposed classification in [4].
2. Create a simulation model from the hydraulic li- brary on the considered level of abstraction.
3. Select the proper design criteria from the para- metric design library and create the static calcu- lation model.
4. Link the requirements model, static calculation model as well as product catalog model in the parametric design model.
5. Run a parameter synthesis to obtain a best suited parameter set and the other possible parameter sets for the desired system.
6. Set the obtained possible parameter sets in the simulation model and save them as design vari- ants.
7. Define test cases that need to satisfy customer re- quirements.
8. Link the requirements model, test cases as well as simulation models in the verification model.
9. Run a verification that executes all related test cases and design variants.
10. Choose the best suited design variants accord- ing to the verification results.

5 Application Example

In this section, a hydraulic lift system is used to demonstrate the static parametric design methodology. The hydraulic lift system is used to lift a load to a given height. It shall be considered in the context of the OEM-supplier relation as it applies to a typical Bosch Rexroth engineering project.

The task of this case study is to define a best suited parameter set of the desired lift system which fulfills all the customers requirements as well as technical constraints. First of all, a simulation model shall be created. Therefore, the structure of the hydraulic lift system has to be known for the system engineers. Then, the static calculation model shall be created as well by selecting the proper design criteria from the parametric design library. After linking of different models in the integrated system model, a parameter synthesis can be performed to obtain the best combination of the components with the minimal dimension which satisfy all the requirements.

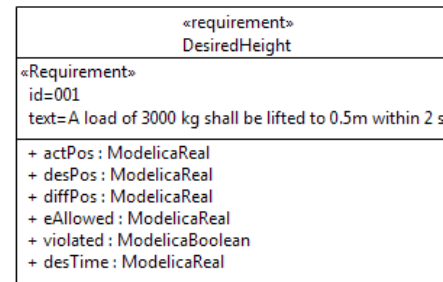
The main advantage is that the system engineers can use the integrated system model as a design template. With the help of this design template, it is much more easier to variate the parameter set of the hydraulic lift system by changing the customer requirements automatically.

5.1 Requirements Capture

The requirements from the customers are formalized as follows: a load of 3000 kg shall be lifted to 0.5 m within 2 s. Besides the customer requirement there are some technical constraints of the desired system. For example, the maximum velocity can not exceed 0.6 m/s. The other important constraint is that the pressure drop over the proportional valve shall not exceed 30% of the working pressure. The important requirements are listed in Table 1. The beginning letter of the ID of the requirement refers to the type of the requirement. The requirements P1, C2 and C3 can be verified by test cases which are modeled as Modelica models. Since the structural requirements S4 and S5 provide only the important design parameters, they are not necessary or possible to verify. According to those requirements, the proper components from the product catalogs shall be selected iteratively until all the components are chosen. They can be formalized as SysML requirements model and later transformed into Modelica static model. An example of the requirements model and its generated Modelica code are shown in Figure 6.

ID	Description
P1	The load shall be lifted to 0.5 m within 2 s.
C2	The max. velocity shall not exceed 0.6 m/s.
C3	The pressure loss over the valve shall not exceed 30% of the working pressure.
S4	The mass of the load is 3000 kg.
S5	The working pressure is 200 bar.

Table 1: The System Requirements List



```

model DesiredHeight
  input Real actPos;
  parameter Real desPos = 0.5;
  parameter Real desTime = 2;
  parameter Real eAllowed=0.005;
  Real diffPos;
  Boolean violated;
end DesiredHeight;

```

Figure 6: An Example of Requirements Model and Generated Modelica Code

5.2 Modelling of the Hydraulic Lift System

The object diagram (Figure 7) shows the structure of the hydraulic lift system. The load is lifted by a differential cylinder which is driven by a constant pressure source. The proportional valve is controlled by a simple P-controller to realize the position control. Since the focus of this work is to illustrate the static parametric design methodology, the details about the model will not be introduced here.

5.3 Static Parametric Design Process

The involved components from the product catalog are the differential cylinder and proportional valve. Hence, the design criteria models for those two components in the design library are selected into the static parametric design model together with the requirements model. Table 2 and 3 show the important design variables from the product catalogs of differential cylinder and proportional valve.

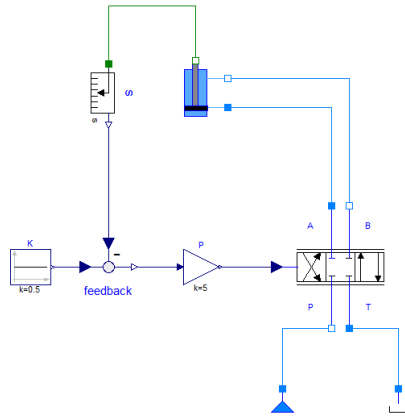


Figure 7: Object Diagram of the Hydraulic Lift System

Piston Diameter mm	Rod Diameter mm	Max Stroke mm
40	28	2000
50	36	2000
63	45	2000
80	56	2000
100	70	3000
125	90	3000
140	100	3000

Table 2: Product Catalog of Differential Cylinders

Nominal Size	Max Flow Rate l/min	Pressure Drop bar
10	170	80
16	450	180
25	900	350
27	1000	430
35	3500	1100

Table 3: Product Catalog of Proportional Valves

The requirements variables defined in the requirements models, such as mass of load, maximum velocity and desired lifting position are taken as inputs for the design criteria which are implemented as Modelica functions. Finally, a parametric design model is obtained by linking the requirements models, static calculation models and the related product catalog models.

After the parametric design model is created, the automatic parameter synthesis can be done very conveniently. The parametric design model is interactive solved and the design variables are calculated. Ac-

ording to those design variables the corresponding components with the proper size are chosen until all the components from the product catalog are chosen. The following table shows the automatic generated best suite combination of cylinder and valve, which is defined as an *optimal design*. It is worth to mention

Cylinder	Piston Diameter mm	Rod Diameter mm	Max Stroke mm
Optimal	100	70	3000
Valve	Nominal Size	Max Flow Rate l/min	Pressure Drop bar
Optimal	25	900	350

Table 4: Selected Components from Product Catalog

that the gain of the P-controller is determined by "Try and Error". In the future, this kind of parameter which is not related to the product catalog can be defined by the means of optimization.

5.4 Verification of System Design

After the static parametric design process is done, a best suited set of combination of the components is obtained. An automatic verification will check the optimal design against customer requirements. This is done by using vVDR methodology [10] to model the test case of requirements with violation monitor. According to the requirements definitions in Table 1, three test cases are defined to verify the requirements P1, C2 and C3. Figure 8 shows the verification result of this proposed optimal design. The first two figures (Figure 8(a) and 8(b)) illustrate that the load is lifted to 0.5 meter after 2 second and does not exceed the maximal velocity. As shown in Figure 8(c), the pressure loss over the valve in steady state satisfies the critical 30% of working pressure 200 bar as well. Therefore, the hydraulic lift system with the automatic selected parameter fulfills the customer requirement and technical constraints.

5.5 Comparison of Design Variants

The best suited combination of components from the product catalog are supposed to have the minimal size which satisfy all the requirements. It has been verified to fulfill all the requirements in the last section. Nevertheless, it is still not proved that the performance of this design is better than the others variants. In order

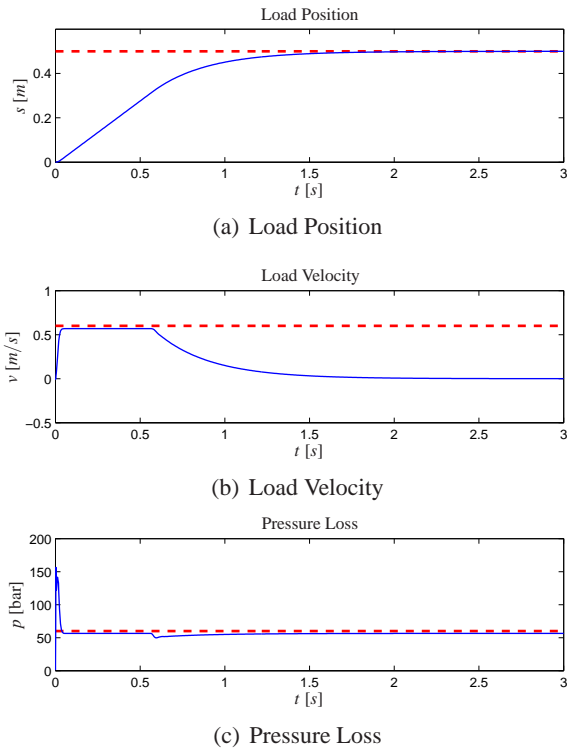


Figure 8: Verification Results of Optimal Design

to validate this, design variants around the optimal design with other nominal sizes can be generated from this methodology. The design variants are defined by substituting the components of the optimal design with a smaller or larger nominal size. In this case, four design variants are automatically generated and used to compare with the optimal design. The dimension of all the design variants are shown in the following table.

Design Variants	Piston Diameter	Rod Diameter	Valve Size
Optimal	100	70	25
Variant 1	80	56	25
Variant 2	100	70	16
Variant 3	100	70	27
Variant 4	125	90	25

Table 5: Dimensions and Costs of Design Variants

The simulation results of the optimal design and the other four design variants are shown in Figure 9. It shows that the design variant 1 and 3 can approach the desired position within 2 seconds. However the velocities exceed the maximal velocity constraint 0.6 m/s. The design variant 2 and 4 fulfill the second test case and can not satisfy the first one. The results concerning the third test case are listed in Table 6.

A verification matrix of the design variants against

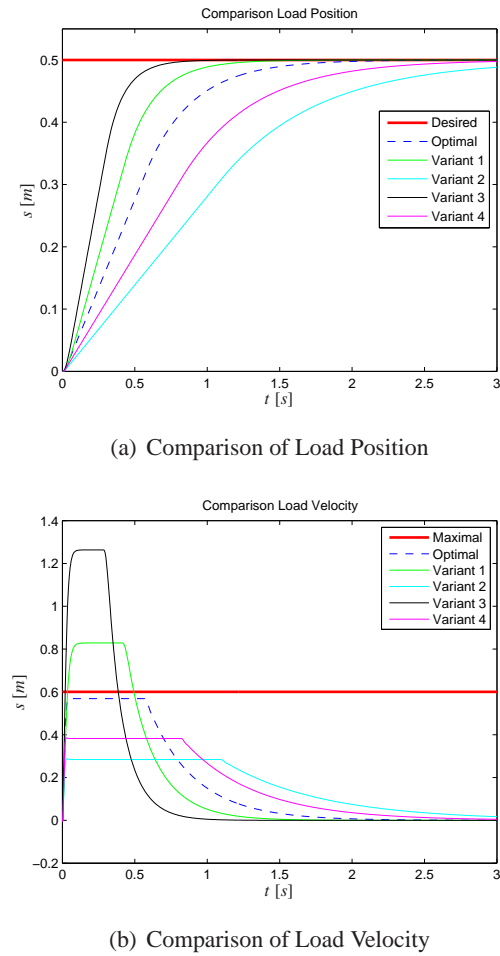


Figure 9: Comparison of Cylinder Positions and Velocities of Different Design Variants

the test cases are shown in Table 6. Furthermore, the costs of design variants depending on the dimensions of selected components from the product catalog can also be calculated. With the help of this verification matrix and the price, the selected optimal combination of the components from the product catalog is proved to be exact the best suited design.

Design Variants	Test Case 1	Test Case 2	Test Case 3	System Cost
Optimal	passed	passed	passed	2000
Variant 1	passed	failed	passed	1700
Variant 2	failed	passed	passed	1820
Variant 3	passed	failed	failed	2040
Variant 4	failed	passed	failed	2400

Table 6: Verification Matrix and System Costs

5.6 Open Issues

This case study demonstrates the proposed static parameter design methodology. According to the customer requirements and technical constraints, the dimension of the desired system can be defined automatically. However, the main drawback is that the simulation model shall be first modeled. That means this methodology can not be applied to arbitrary system. This is due to the fact that there is not enough information for determining a proper combination of the desired system in the practice. This drawback also limit the application of MBSE in the field of industrial automation systems.

It is noticed that the order for the selection of components is fixed in this case, i.e., the dimension of cylinder shall be first defined in order to determine the nominal size of the proportional valve. Sometimes the order is not fixed. For both cases the system engineers shall have the chance to determine the order for the selection of components more freely without reimplementation of static design model. Since Modelica is a standardize equation-based modelling language, it has been chosen to meet these requirements. It is capable to deal with the this issue. One proposed concept is to switch the variability of *parameter* and *variable* of static models in an arbitrary manner. The system engineers can give the known parameters until the static model is balanced and solvable to calculate the other unknown variables.

6 Conclusion and Future Work

In this paper a static parametric design methodology has been analyzed in the systems engineering context of industrial automation systems. A set of possible design variants with different dimensions can be automatically generated and compared by using this methodology. The concept has been demonstrated by a case study of a typical engineering project. The other contribution of this work is allocation of this methodology in a MBSE process in which the parametrized design variants are fully traceable to the other models.

In the future, the proposed methodology will be implemented as an Eclipse plug-in for better tool support of the static parametric design. It is usually the case, not all the parameters can be defined by the static parametric design methodology. Integration of an optimizer to define those parameters is desired. Application of a big scenario is also a part of future work.

Acknowledgments

This work is funded by Bosch Rexroth AG and German Federal Ministry of Education and Research (BMBF) in the ITEA2 OPENPROD project.

References

- [1] <http://www.acceleo.org>.
- [2] Hubert Dubois, Marie-Agnès Peraldi-Frati, and Fadoi Lakhal. A Model for Requirements Traceability in a Heterogeneous Model-Based Design Process: Application to Automotive Embedded Systems. In *15th IEEE International Conference on Engineering of Complex Computer Systems, ICECCS 2010, Oxford, United Kingdom, 22-26 March 2010*, pages 233–242, 2010.
- [3] Martin Glinz. On Non-Functional Requirements. In *15th IEEE International Requirements Engineering Conference, RE 2007, October 15-19th, 2007, New Delhi, India*, pages 21–26, 2007.
- [4] Hongchao Ji, Oliver Lenord, and Dieter Schramm. A Model Driven Approach for Requirements Engineering of Industrial Automation Systems. In *Proceedings of the 4th International Workshop on Equation-Based Object-Oriented Languages and Tools (EOOLT'07)*, pages 13–24, 2011.
- [5] Thomas Johnson, Christiaan J.J. Paredis, and Roger Burkhart. Integrating Models and Simulations of Continuous Dynamics into SysML. In *Proceedings of 6th International Modelica Conference, Bielefeld, Germany, 3-4, March, 2008*, 2008.
- [6] Christiaan J.J. Paredis, Yves Bernard, Roger M. Burkhart, Hans-Peter de Koning, Sanford Friedenthal, Peter Fritzson, Nicolas F. Rouquette, and Wladimir Schamai. An Overview of the SysML-Modelica Transformation Specification. In *2010 INCOSE International Symposium*, July 2010.
- [7] Adrian Pop, David Akhvlediani, and Peter Fritzson. Towards Unified System Modeling with the ModelicaML UML Profile. In *Proceedings of the 1st International Workshop on Equation-Based Object-Oriented Languages and Tools (EOOLT'07)*, pages 13–24, 2007.

- [8] Wladimir Schamai. Modelica Modeling Language (ModelicaML). Technical report, EADS Innovation Works, Germany, 2009.
- [9] Wladimir Schamai, Peter Fritzson, Chris Paredis, and Adrian Pop. Towards Unified System Modeling and Simulation with ModelicaML: Modeling of Executable Behavior Using Graphical Notations. In *Proceedings of the 7th International Modelica Conference, Como, Italy, 20-22 September 2009*, number 43 in Linköping Electronic Conference Proceedings, pages 612–621. Linköping University Electronic Press, Linköpings universitet, December 2009.
- [10] Wladimir Schamai, Philipp Helle, Peter Fritzson, and Christiaan J. J. Paredis. Virtual Verification of System Designs against System Requirements. In *Models in Software Engineering - Workshops and Symposia at MODELS 2010, Oslo, Norway, October 2-8, 2010, Reports and Revised Selected Papers*, pages 75–89, 2010.
- [11] <http://www.omg.sysml.org>.
- [12] VDI. Design Methodology for Mechatronic Systems (VDI 2206). Technical report, VDI, 2004.

Exhibitors

BAUSCH-GALL GmbH

BAUSCH-GALL GmbH

BAUSCH-GALL GmbH (LLC) is an engineering company based in Munich, Germany, which sells and supports Modelica Libraries, works on simulation projects, organizes training courses and does consulting based on specific technical know-how. BAUSCH-GALL GmbH also offers special design services, devices and products for radio frequency (RF) applications. Based on a broad range of expertise in the solution of practical problems by effective computer application, BAUSCH-GALL GmbH serves the market for simulation and computer-aided engineering.

CENIT



DRIVEN BY YOUR VISION.

CENIT has been successfully active for more than 20 years as a leading consulting and software specialist for optimizing business processes in product lifecycle management (PLM), enterprise information management (EIM), business optimization & analytics (BOA) and application management services (AMS). The enterprise focuses chiefly on proprietary software development and on marketing standard solutions by market leaders such as Dassault Systèmes, SAP and IBM. CENIT employs about 700 staff world-wide, serving customers from the automotive, aerospace, mechanical engineering, tool and mold construction, financial services, commercial and consumer goods industries.

Claytex



Claytex is an engineering consultancy and software distributor that specialises in Systems Engineering. Our expertise is in the modelling and simulation of complex multi-domain systems using Dymola and Modelica. We are based in Leamington Spa (UK) and work with Modelica and FMI on a wide variety of projects. Most recently these include the modelling of Low Carbon Vehicles, Formula 1 and Nascar Sprint Cup racing cars. These projects apply the models in a wide range of tasks including energy usage calculations, control system development, powertrain design and driving simulators. We develop a number of application libraries for Dymola include the Engines, Powertrain Dynamics, SystemID, FlexBody, VDLMotorsports and XMLReader libraries.

CyDesign Labs



CyDesign Labs focuses on the conceptual design phase of complex cyber-electromechanical systems. CyDesign is a model-based design optimization platform that will allow system designers to evaluate a broad range of alternative designs and verify major system requirements prior to detailed design. The platform uses Modelica to simulate system behavior for requirements verification. A comprehensive component model library allows designers to concentrate on strategic design decisions instead of model development. By thorough exploration of the trade space of design alternatives and assessment of the viability of these alternatives, repetitive “design-build-test” cycles can be eliminated, resulting in reduced costs and time to market. The CyDesign platform will be available for automotive applications starting in early 2013.

Dassault Systèmes



Dassault Systèmes, the 3DEXPERIENCE Company, provides business and people with virtual universes to imagine sustainable innovations. Its world-leading solutions transform the way products are designed, produced, and supported. Dassault Systèmes’ collaborative solutions foster social innovation, expanding possibilities for the virtual world to improve the real world. The group brings value to over 150,000 customers of all sizes in all industries in more than 80 countries.

Dassault Systèmes’ CATIA provides a fully integrated systems modeling environment that enables systems engineers to execute and analyze system or sub-systems models, while mixing dynamic and state logic behaviors, using the open source Modelica language.

ITI GmbH



Supporting your vision

In the realm of system simulation, ITI is a leading developer of innovative software solutions and offers a vast range of engineering services that help to reduce time-to-market significantly. Our interdisciplinary software application SimulationX allows for comprehensive physical modeling of complex systems. Amongst others we support our customers in virtual prototyping, result interpretation and optimization of energy-efficient design. SimulationX supports the Modelica® language with open and complete CAx interfaces. The software is applied by more than 700 well-known companies, such as Audi, BMW, Bureau Veritas, Daimler, Fraunhofer-Gesellschaft, Germanischer Lloyd, Honda, Nikon, Robert Bosch, Siemens, ThyssenKrupp und Veolia.

LMS International



Since the 2010 conference, LMS considerably increased its effort to make Imagine.Lab the best-of-breed platform for system simulation. LMS' will is to deliver a combined structured approach (C-based and Modelica-based) to best serve the engineering needs, from full system to detailed component modeling over most of the mechatronics applications. LMS continues to support the establishment of Modelica as an industrial reference through its dedicated commercial support team as well as its involvement in European research projects and its support of the FMI. LMS's position in the Model Based System Engineering software market is considerably increasing to the benefit of the industry and the recognition of Modelica.

MapleSoft



Maplesoft, a subsidiary of Cybernet Systems Co., Ltd. in Japan, is the leading provider of high-performance software tools for engineering, science, and mathematics. Its product suite reflects the philosophy that given great tools, people can do great things. Maplesoft's core technologies include Maple, the world's most advanced symbolic computation engine, and MapleSim, a Modelica-based physical modeling and simulation tool. With MapleSim, you can leverage the growing collection of industry-tested Modelica components in your own projects. Maplesoft's customers include Ford, BMW, Bosch, Boeing, NASA, CSA, Canon, Motorola, Microsoft, Bloomberg, and DreamWorks, covering sectors such as automotive, aerospace, electronics, defense, and energy.

Modelon GmbH



Modelon specializes in providing solutions, services and technology for the research and development of dynamic systems. We offer unique know-how in physical modeling, simulation and optimization, and model-based control design. Our customers are spread all over the world and represent a variety of application areas with some emphasis on the automotive, energy and process industries. Modelon has a strong competitive edge in technology and solutions related to the Modelica language, takes active part in the language development, and is the premier provider of commercial Modelica libraries. Besides offering engineering services and modelica libraries, Modelon provides Dymola Training Days for beginners and advanced users as well. The company has sites in Germany, Sweden and also in USA.

Open Modelica

OpenModelica

OpenModelica is an open-source Modelica-based modeling and simulation environment intended for industrial and academic usage. Its long-term development is supported by a non-profit organization – the Open Source Modelica Consortium (OSMC). The goal with the OpenModelica effort is to create a comprehensive Open Source Modelica modeling, compilation and simulation environment based on free software distributed in binary and source code form for research, teaching, and industrial usage. We invite researchers and students, or any interested developer to participate in the project and cooperate around OpenModelica, tools, and applications.

QTronic GmbH



QTronic provides engineering software and services for model-based development.

- Silver is a tool used to move control development tasks from real cars, test rigs and HiLs to Windows PC. Silver provides an environment to quickly port control software (C Code or hex file) from real ECUs to Windows and to run the resulting virtual ECU in closed-loop with a simulated vehicle on PC.
- TestWeaver automates search for worst case system behavior, based on an executable system model and system quality indicators. The objective is to early identify bugs and weak points of a system via MiL/SiL/HiL.

Our outstanding tools are used by development engineers at AMG, BMW, Bosch, Continental, Honda, IAV, Mercedes-Benz, Toyota and ZF.

Schlegel Simulation GmbH



Schlegel Simulation GmbH is an engineering company and software distributor based in Munich, Germany. Our expertise is modeling and simulation of mechatronic systems using Dymola / Modelica and other tools. We develop simulation models, work on simulation projects, realtime and hardware-in-the-loop simulations, we develop customer specific simulators and software, and provide consultancy and training. Schlegel Simulation distributes and supports Dymola.

SIMPACK AG



Leading MBS Technology
for Technology Leaders

SIMPACK AG, a spin-off of the DLR, was founded in 1993. The company expanded quickly in the sectors of Virtual Prototyping and 3D-Simulation and soon achieved international recognition for excellence.

SIMPACK is used for non-linear multi-body simulation and is particularly renowned for the integration of flexible bodies. The simulation of cars, trucks, engines, rail vehicles, wind turbines and airplanes represent only some sectors where SIMPACK is used. SIMPACK is the market leader in the simulation of high frequency vibrations and 'shock contact' and therefore the number one choice for the handling and comfort analyses as well as NVH and durability calculations. The software's diversity and good connectivity to various CAD, control, hydraulic and FE software enables SIMPACK to be easily integrated into any manufacturer's already established development process.

TLK-Thermo GmbH



TLK-Thermo GmbH has a long experience in R&D with a focus on energy management, mobile air conditioning and refrigeration systems. During the last years TLK has been continuously increasing the scope of its activities on other thermal systems such as power plants, residential heating and industrial refrigeration systems and the thermal management of alternative vehicle concepts. TLK provides its expertise in thermodynamics, simulation technology and software as engineering services. We offer simulation and measurement of thermal systems, customized software, consulting and training courses. Our software products are TIL Suite/TILMedia Suite (modeling of thermal systems), TISC Suite (Co-Simulation environment), FMI Suite und ViewerSuite.

Transcat PLM GmbH



Founded in 1987, Transcat PLM GmbH is an established specialist and supplier of Product Lifecycle Management (PLM) solutions based on the CATIA, ENOVIA, DELMIA, SIMULIA and 3DVIA products of Dassault Systèmes. As a Value Added Reseller (VAR) the company offers the PLM solutions throughout Germany with the associated services as well as its own add-on software products. As one of a few partners Transcat PLM is certificated for all V6 products of the PLM 2.0 portfolio and markets the complete V6-product range of Dassault Systèmes. Transcat PLM also offers customized software components for virtual product development in its Product Data Quality (PDQ) range. The portfolio is extended by tailored server, storage and system management concepts.

Wolfram



Founded by Stephen Wolfram in 1987, Wolfram is one of the world's most respected software companies. At the center is Mathematica: the world's most powerful global computation system. In 2011, Wolfram acquired MathCore Engineering AB - a founding member of the Modelica Association and an active influence in the Modelica language design since 1997. Through this, SystemModeler was released in 2012 - the most complete physical modeling and simulation tool. Unlike other systems, SystemModeler requires no add-ons, fully supports the standard Modelica model language and is designed to connect perfectly with Mathematica for the ultimate integrated modeling, simulation, and analysis workflow.

XRG Simulation GmbH



XRG Simulation has extended expertise in thermal energy system simulations in the automotive and building services field, for the aerospace and shipping industry and for power plants. We are specialized in energy engineering and support industry and research institutions in research, development and improvement of products and projects. Our excellence is:

- Modelling and simulation of thermodynamic systems
- Mathematical optimization
- Validation of models
- Software development for optimization as well as pre- and post-processing of system simulations