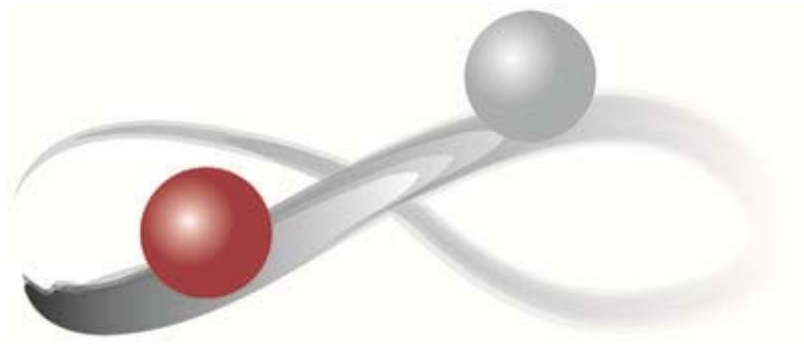


Proceedings of the 8th International Modelica Conference

March 20th–22nd, 2011
Auditorium Centre of the Technische Universität
Dresden, Germany



Christoph Clauß (editor)

Organized by
Modelica Association and the Fraunhofer IIS EAS

Proceedings of the 8th Modelica Conference

Auditorium Centre of the Technische Universität Dresden, Germany
March 20th–22nd, 2011

Editor:

Christoph Clauß

Published by:

The Modelica Association and Linköping University Electronic Press

ISBN: 978-91-7393-096-3

Linköping Electronic Conference Proceedings

ISSN: 1650-3740

DOI: <http://dx.doi.org/10.3384/ecp11063>

Copyright © The Modelica Association, 2011

Preface

From March 20th to 22nd 2011 the 8th international Modelica Conference took place in the Auditorium Center of the Technische Universität in Dresden.

Modelica is an object oriented, physical modelling language, which allows a very effective description of different physical systems. It has been developed for the last 15 years. The last Modelica Conferences, starting in Lund in 2000, via Oberpfaffenhofen, Linköping, Hamburg-Harburg, Vienna, Bielefeld, Como and now Dresden show, that the language is used in an increasing area of applications in industry, research and education. The 8th international Modelica Conference in Dresden was with its 316 participants from 23 different countries the biggest Modelica Conference until now.

Besides 76 oral presentations and 23 poster presentations also the possibility of interesting discussions to socialize with new people was given. Tutorials, to go more into detail into about single topics in the Modelica area, were visited on Sunday, 20th of March from already about 100 people. In the evening already 200 people joined the “Coming Together”.

On Monday the Conference was opened by the sitting mayor of Dresden, Mr. Dirk Hilbert, followed by greetings from Prof. Elst, head of the Fraunhofer IIS EAS and Prof. Martin Otter, chairman of the Modelica Association. Keynote speeches were held by Dr. Peter Schwarz, formerly department manager at Fraunhofer IIS EAS, about the requirements on simulation of complex heterogeneous systems and by Scott A. Borthoff from the Mitsubishi Electric Research Laboratories Cambridge (USA) about his experiences with the Modelica language for the last decade and future challenges.

The topics in the Conference of course were the language itself and its further development. Application-oriented papers were presented in the fields of automotive, mechanics and fluidic, power plants, electrical engineering, as well as in new opened application fields like building, climate, and biology. Many papers covered the Functional Mockup Interface (FMI) which is an interface for the coupling of simulators as well as for the model export to support the exchange of models between different simulation environments. The papers reviewed to be the best ones described the application of homotopy methods to improve the initialization phase of simulations, using the recently introduced homotopy operator.

The small exhibition was an important conference issue. 14 tool vendors as well as companies which use Modelica for modelling tasks presented their products. Another important conference issue for communication was the conference dinner at “The Westin Bellevue” hotel. The conference dinner was sponsored by ITI GmbH Dresden.

Dresden, April 1, 2011

Dr. Christoph Clauß
Program Chair

Organizing Committees

Program Chair

- Dr. Christoph Clauß, Fraunhofer IIS EAS, Dresden, Germany

Program Board

- Dr. Hilding Elmqvist, Dassault Systèmes, Lund, Sweden
- Prof. Peter Fritzson, Linköping University, Linköping, Sweden
- Prof. Martin Otter, DLR, Oberpfaffenhofen, Germany
- Dr. Michael Tiller, Emmeskay Inc., Plymouth, Michigan, USA

Program Committee

- Dr. Johan Åkesson, Lund University, Lund, Sweden
- Prof. Karl-Erik Årzén, Lund University, Lund, Sweden
- Prof. Bernhard Bachmann, University of Applied Sciences Bielefeld, Bielefeld, Germany
- Dr. John Batteh, Emmeskay Inc., Plymouth, Michigan, USA
- Dr. Ingrid Bausch-Gall, Bausch-Gall GmbH, Munich, Germany
- Daniel Bouskela, EDF R&D, Paris, France
- Prof. Francesco Casella, Politecnico di Milano, Milano, Italy
- Prof. François E. Cellier, ETH Zürich, Zürich, Switzerland
- Mike Dempsey, Claytex Services Ltd, Leamington Spa, UK
- Prof. Gianni Ferretti, Politecnico di Milano, Milano, Italy
- Dr. Rüdiger Franke, ABB AG, Mannheim, Germany
- Dr. Rui Gao, Dassault Systèmes Japan, Nagoya, Japan
- Anton Haumer, Technical Consultant, St. Andrae-Woertern, Austria
- Dr. Kay Juslin, VTT, Espoo, Finland
- Dr. Christian Kral, AIT Arsenal Research, Vienna, Austria
- Prof. Alberto Leva, Politecnico di Milano, Milano, Italy
- Kilian Link, Siemens AG, Erlangen, Germany
- Dr. Sven-Erik Mattsson, Dassault Systèmes, Lund, Sweden
- Dr. Jakob Mauss, QTronic GmbH, Berlin, Germany
- 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. Adrian Pop, Linköping University, Linköping, Sweden
- Prof. Gerhard Schmitz, Technical University Hamburg-Harburg, Hamburg, Germany
- Dr. Peter Schneider, Fraunhofer IIS, Institutsteil EAS, Dresden, Germany
- Dr. Edward D. Tate, General Motors, Detroit, Michigan, USA
- Dr. Wilhelm Tegethoff, TLK-Thermo GmbH and TU Braunschweig, Braunschweig, Germany
- Dr. Hubertus Tummescheit, Modelon AB, Lund, Sweden
- Dr. Andreas Uhlig, ITI GmbH, Dresden, Germany
- Prof. Alfonso Urquía, UNED, Madrid, Spain
- Prof. Hans Vangheluwe, McGill University, Montreal, Canada
- Dr. Hansjürg Wiesmann, formerly ABB Corporate Research, Baden, Switzerland

Local Organizing Committee

- Sibylle Läßig, Fraunhofer IIS EAS, Dresden, Germany
- Kristin Majetta, Fraunhofer IIS EAS, Dresden, Germany
- André Schneider, Fraunhofer IIS EAS, Dresden, Germany
- Dr. Christoph Clauß, Fraunhofer IIS EAS, Dresden, Germany

Contents

Emmanuel Chrisofakis, Andreas Junghanns, Christian Kehrer, Anton Rink: <i>Simulation-based development of automotive control software with Modelica</i>	1
Edo Drenth, Magnus Gäfvert: <i>Modelica Delft-Tyre Interface</i>	8
Andreas Deuring, Johannes Gerl, Harald Wilhelm: <i>Multi-Domain Vehicle Dynamics Simulation in Dymola</i>	13
Dietmar Winkler, Hege Marie Thoresen, Ingvar Andreassen, Magamage Anushka Sampath Perera, Rahimi Behzad Sharefi: <i>Modelling and Optimisation of Deviation in Hydro Power Production</i>	18
Karin Dietl, Kilian Link, Gerhard Schmitz: <i>Thermal Separation Library: Examples of Use</i>	28
Marco Bonvini, Alberto Leva: <i>Scalable-detail modular models for simulation studies on energy efficiency</i>	39
Christian Schulze, Manuel Gräber, Michaela Huhn, Uwe Grätz: <i>Real-Time Simulation of Vapour Compression Cycles</i>	48
Alberto Leva, Marco Bonvini: <i>Efficient hybrid simulation of autotuning PI controllers</i>	56
J. Åkesson, R. Faber, C. D. Laird, K. Pröhl, H. Tummescheit, S. Velut, Y. Zhu: <i>Models of a post-combustion absorption unit for simulation, optimization and non-linear model predictive control schemes</i>	64
Michael Sielemann, Francesco Casella, Martin Otter, Christoph Clauß, Jonas Eborn, Sven Erik Mattsson, Hans Olsson: <i>Robust Initialization of Differential-Algebraic Equations Using Homotopy</i>	75
Francesco Casella, Michael Sielemann, Luca Savoldelli: <i>Steady-state initialization of object-oriented thermo-fluid models by homotopy methods</i>	86
Johan Ylikiiskilä, Johan Åkesson, Claus Führer: <i>Improving Newton's method for Initialization of Modelica models</i>	97
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: <i>The Functional Mockup Interface for Tool independent Exchange of Simulation Models</i>	105
Jens Bastian, Christoph Clauß, Susann Wolf, Peter Schneider: <i>Master for Co-Simulation Using FMI</i>	115
Andreas Heckmann, Stefan Hartweg, Ingo Kaiser: <i>An Annular Plate Model in Arbitrary Lagrangian-Eulerian Description for the DLR FlexibleBodies Library</i>	121
Damien Chapon, Fabien Hospital, Guillaume Bouchez, Marc Budinger: <i>A Modelica-Based and Domain-Specific Framework for Electromechanical System Design</i>	133
Jens Frenkel, Christian Schubert, Günter Kunze, Peter Fritzson, Martin Sjölund, Adrian Pop: <i>Towards a Benchmark Suite for Modelica Compilers: Large Models</i>	143
Mohsen Torabzadeh-Tari, Zoheb Muhammed Hossain, Peter Fritzson, Thomas Richter: <i>OMWeb – Virtual Web-based Remote Laboratory for Modelica in Engineering Courses</i>	153
Kristin Majetta, Sandra Böhme, Christoph Clauß, Peter Schneider: <i>MSL Electrical Spice3 - Status and Further Development</i>	160

Christian Kral, Anton Haumer: <i>The New FundamentalWave Library for Modeling Rotating Electrical Three Phase Machines</i>	170
Tilman Bunte, Emmanuel Chrisofakis: <i>A Driver Model for Virtual Drivetrain Endurance Testing</i>	180
Torsten Schwan, René Unger, Bernard Bäker, Beate Mikoleit, Christian Kehrer: <i>Optimization-Tool for local renewable energy usage in the connected system: "Building-eMobility"</i>	189
Thomas Kaden, Klaus Janschek: <i>Development of a Modelica Library for Simulation of Diffractive Optomechatronic Systems</i>	199
Kerstin Bauer, Klaus Schneider: <i>Transferring Causality Analysis from Synchronous Programs to Hybrid Programs</i>	207
Joel Andersson, Johan Åkesson, Francesco Casella, Moritz Diehl: <i>Integration of CasADi and JModelica.org</i>	218
Jens Frenkel, Günter Kunze, Peter Fritzon, Martin Sjölund, Adrian Pop, Willi Braun: <i>Towards a Modular and Accessible Modelica Compiler Backend</i>	232
Chen Chang, Ding Jianwan, Chen Liping, Wu Yizhong: <i>Media Modeling for Distillation Process with Modelica/Mworks</i>	239
Martin Ryhl Kærn, Brian Elmegaard, Lars Finn Sloth Larsen: <i>Experimental comparison of the dynamic evaortor response using homogeneous and slip flow modeling</i>	246
Antoine Viel: <i>Strong Coupling of Modelica System-Level Models with Detailed CFD Models for Transient Simulation of Hydraulic Components in their Surrounding Environment</i>	256
Michael Wetter, Wangda Zuo, Thierry Stephane Nouidui: <i>Recent Developments of the Modelica "Buildings" Library for Building Energy and Control Systems</i>	266
Marco Bonvini, Alberto Leva: <i>Object-oriented sub-zonal room models for energy-related building simulation</i>	276
Manuel Ljubijankic, Christoph Nytsch-Geusen, Jörg Rädler, Martin Löffler: <i>Numerical coupling of Modelica and CFD for building energy supply systems</i>	286
José Luis Reyes Péres Andreas Heckmann, Ingo Kaiser: <i>A Thermo-elastic Annular Plate Model for the Modeling of Brake Systems</i>	295
Volker Beuter: <i>An Interface to the FTire Tire Mode</i>	304
Ivan Kosenko, Il'ya Gusev: <i>Implementation of the Spur Involute Gear Model on Modelica</i>	315
Christian Andersson, Johan Åkesson, Claus Führer, Magnus Gäfvert: <i>Import and Export of Functional Mock-up Units in JModelica.org</i>	329
Christian Noll, Torsten Blochwitz, Thomas Neidhold, Christian Kehrer: <i>Implementation of Modelisar Functional Mock-up Interfaces in SimulationX</i>	339
Christian Schubert, Thomas Neidhold, Günter Kunze: <i>Experiences with the new FMI Standard Selected Applications at Dresden University</i>	344
Sebastian Meinke, Friedrich Gottelt, Martin Müller, Egon Hassel: <i>Modeling of Coal-Fired Power Units with ThermoPower Focussing on Start-Up Process</i>	353
Baligh El Hefni, Daniel Bouskela, Grégory Lebreton: <i>Dynamic modelling of a combined cycle power plant with ThermoSysPro</i>	365
Daniel Rene Kreuzer, Andreas Werner: <i>Implementation of Models for reheating processes in industrial furnaces</i>	376

Anton Haumer, Christian Kral: <i>Modeling a Mains connected PWM Converter with Voltage-Oriented Control</i>	388
Jonathan Brembeck, Sebastian Wielgos: <i>A real time capable battery model for electric mobility applications using optimal estimation methods</i>	398
Oliver Chilard, Jean-Philippe Tavella, Olivier Devaux: <i>Use of Modelica language to model an MV compensated electrical network and its protection equipment: comparison with EMTP</i>	406
Johan Andreasson: <i>The Vehicle Dynamics Library: New Concepts and New Fields of Application</i>	414
Chen Liping, Zhao Yan, Zhou Fanli, Zhao Jianjun, Tian Xianzhao: <i>Modeling and Simulation of Gear Pumps based on Modelica/MWorks®</i>	421
Li He, Xiaolong Wang, Yunqing Zhang, Jinglai Wu, Liping Chen: <i>Modeling and Simulation Vehicle Air Brake System</i>	430
M. Einhorn, F. V. Conte, C. Kral, C. Niklas, H. Popp, J. Fleig: <i>A Modelica Library for Simulation of Electric Energy Storages</i>	436
Stéphane Velut, Hubertus Tummescheit: <i>Implementation of a transmission line model for fast simulation of fluid flow dynamics</i>	446
Kilian Link, Stephanie Vogel, Ines Mynttinen: <i>Fluid Simulation and Optimization using Open Source Tools</i>	454
Christian Bayer, Olaf Enge-Rosenblatt: <i>Modeling of hydraulic axial piston pumps including specific signs of wear and tear</i>	461
Sebastian Nowoisky, Chi Shen, Clemens Gühmann: <i>Detailed Model of a Hydro-mechanical Double Clutch Actuator with a Suitable Control Algorithm</i>	467
Jonatahan Brembeck, Martin Otter, Dirk Zimmer: <i>Nonlinear Observers based on the Functional Mockup Interface with Applications to Electric Vehicles</i>	474
Bernhard Thiele, Dan Henriksson: <i>Using the Functional Mockup Interface as an Intermediate Format in AUTOSAR Software Component Development</i>	484
Yongqi Sun, Stephanie Vogel, Haiko Steuer: <i>Combining Advantages of Specialized Simulation Tools and Modelica Models using Functional Mock-up Interface (FMI)</i> ..	491
Willi Braun, Lennart Ochel, Bernhard Bachmann: <i>Symbolically Derived Jacobians Using Automatic Differentiation – Enhancement of the OpenModelica Compiler</i>	495
Dan Henriksson, Hilding Elmqvist: <i>Cyber-Physical Systems Modeling and Simulation with Modelica</i>	502
Martin Sjölund, Peter Fritzson, Adrian Pop: <i>Bootstrapping a Modelica Compiler aiming at Modelica 4</i>	510
Daniel Schlabe, Tobias Knostmann, Tilman Bunte: <i>A Scade Suite to Modelica Interface</i>	522
Roland Samlaus, Claudio Hillmann, Birgit Demuth, Martin Krebs: <i>Towards a model driven Modelica IDE</i>	528
Michaela Huhn, Martin Sjölund, Wuzhu Chen, Christian Schulze, Peter Fritzson: <i>Tool Support for Modelica Real-time Models</i>	537
Lionel Belmon, Chen Liu: <i>High-speed train pneumatic braking system with wheel-slide protection device: A modeling application from system design to HIL testing</i>	549
Sabrina Proß, Bernhard Bachmann: <i>An Advanced Environment for Hybrid Modeling and Parameter Identification of Biological Systems</i>	557

A. Bader, S. Bauersfeld, C. Brunhuber, R. Pardemann, B. Meyer: <i>Modelling of a Chemical Reactor for Simulation of a Methanisation Plant</i>	572
Audrey Jardin, Daniel Bouskela, Thuy Nguyen, Nancy Ruel, Eric Thomas, Laurent Chastanet, Raphaël Schoenig, Sandrine Loembé: <i>Modelling of System Properties in a Modelica Framework</i>	579
Daniel Andersson, Erik Åberg, Jonas Eborn, Jinliang Yuan, Bengt Sundén: <i>Dynamic modeling of a solid oxide fuel cell system in Modelica</i>	593
M. Strobel, F. Vorpahl, C. Hillmann, X. Gu, A. Zuga, U. Wihlfahrt: <i>The OnWind Modelica Library for Offshore Wind Turbines – Implementation and first results</i>	603
Leo Gall, Kilian Link, Haiko Steuer: <i>Modeling of Gas-Particle-Flow and Heat Radiation in Steam Power Plants</i>	610
Christian Schallert: <i>Inclusion of Reliability and Safety Analysis Methods in Modelica</i>	616
Stephan Seidel, Ulrich Donath: <i>Error-free Control Programs by means of Graphical Program Design, Simulation-based Verification and Automatic Code Generation</i>	628
Sébastien Furic: <i>Enforcing Reliability of Discrete-Time Models in Modelica</i>	638
Peter Harman: <i>Effective Version Control of Modelica Models</i>	650
Xenofon Floros, Federico Bergero, François E. Cellier, Ernesto Kofman: <i>Automated Simulation of Modelica Models with QSS Methods: The Discontinuous Case</i>	657
The-Quan Pham, Alfred Kamusella, Holger Neubert: <i>Auto-Extraction of Modelica Code from Finite Element Analysis or Measurement Data</i>	668
Daniel Bouskela, Audrey Jardin, Zakia Benjelloun-Touimi, Peter Aronsson Peter Fritzson: <i>Modelling of Uncertainties with Modelica</i>	673
Tilman Bunte: <i>Recording of Model Frequency Responses and Describing Functions in Modelica</i>	686
Posters	
Anton Sodja, Borut Zupančič: <i>On using model approximation techniques for better understanding of models implemented in Modelica</i>	697
Tolga Kurtoglu, Peter Bonus, Johan De Kleer, Rahul Rai: <i>Simulation-Based Design of Aircraft Electrical Power Systems</i>	704
Jiri Kofranek, Marek Matejak, Pavol Privitzer: <i>HumMod - Large Scale Physiological Models in Modelica</i>	713
Ying Sun, Wie Chen, Yunqing Zhang, Liping Chen: <i>Modeling and Simulation of Heavy Truck with MWorks</i>	725
Bart Verbruggen, Juan Van Roy, Roel De Coninck, Ruben Baetens, Lieve Helsen, Johan Driesen: <i>Object-Oriented Electrical Grid and Photovoltaic system modelling in Modelica</i>	730
Syed Adeel Asghar, Sonia Tariq, Mohsen Torabzadeh-Tari, Peter Fritzson, Adrian Pop, Martin Sjölund, Parham Vasaiely, Wladimir Schamai: <i>An Open Source Modelica Graphic Editor Integrated with Electronic Notebooks and Interactive Simulation</i>	739
Olaf Enge-Rosenblatt, Christoph Clauß, André Schneider, Peter Schneider: <i>Functional Digital Mock-up and the Functional Mock-up Interface – Two Complementary Approaches for a Comprehensive Investigation of Heterogeneous Systems</i>	748

Hubert Thieriot, Maroun Nemera, Mohsen Torabzadeh-Tari, Peter Fritzson, Peter, Rajiv Singh, John John Kocherry: <i>Towards Design Optimization with OpenModelica Emphasizing Parameter Optimization with Genetic Algorithms</i>	756
Zakia Benjelloun-Touimi, Mongi Ben Gaid, Julien Bohbot, Alain Dutoya, Hassan Hadj-Amor, Philippe Moulin, Housseem Saafi, Nicolas Pernet: <i>From Physical Modeling to Real-Time Simulation : Feed back on the use of Modelica in the engine control development toolchain</i>	763
Oscar Duarte: <i>UN-VirtualLab : A web simulation environment of OpenModelica models for educational purposes</i>	773
Stephan Ziegler, Robert Höpler: <i>Extending the IPG CarMaker by FMI Compliant Units.</i>	779
Vincenzo Manzoni, Francesco Casella: <i>Minimal Equation Sets for Output Computation in Object-Oriented Models</i>	784
Vlastimil Votrubec, Pavel Šidlof: <i>Optimization example of industrial sewing machines mechanisms</i>	791
Clemens Schlegel, Reinhard Finsterwalder: <i>Automatic Generation of Graphical User Interfaces for Simulation of Modelica Models</i>	796
Mohsen Torabzadeh-Tari, Martin Sjölund, Adrian Pop, Peter Fritzson: <i>DrControl — An Interactive Course Material for Teaching Control Engineering</i>	801
Jörg Frochte: <i>Modelica Simulator Compatibility - Today and in Future</i>	812
Oscar Duarte: <i>A Modelica model of thermal and mechanical phenomena in bare overhead conductors</i>	819
Ming Jiang, Jiangang Zhou, Wei Chen, Yunqing Zhang, Liping Chen: <i>Modeling and Simulation of AMT with MWorks</i>	829
Junjie Tang, Jianwan Ding, Liping Chen, Xiong Gong: <i>Variability and Type Calculation for Equation of Modelica model</i>	837
Yuming Hou, Lingyang Li, Ping He, Yunqing Zhang, Liping Chen: <i>Shock Absorber Modeling and Simulation Based on Modelica</i>	843
Ruben Baetens, Dirk Saelens: <i>Integrating occupant behaviour in the simulation of coupled electric and thermal systems in buildings</i>	847
Ingela Lind, Henric Andersson: <i>Model Based Systems Engineering for Aircraft Systems – How does Modelica Based Tools Fit?</i>	856
Emerson Jacob Jeganathan, Anand Pitchaikani, Elavarasan Dharumaseelan: <i>Productivity improvement tool for configuration of Modelica plant models and integration with Simulink controller models</i>	865

Simulation-based development of automotive control software with Modelica

Emmanuel Chrisofakis¹, Andreas Junghanns², Christian Kehrer³, Anton Rink¹

¹Daimler AG, 70546 Stuttgart

²QTronic GmbH, Alt-Moabit 91a, 10559 Berlin

³ITI GmbH, Webergasse 1, 01067 Dresden

{emmanuel.chrisofakis, anton.rink}@daimler.com, andreas.junghanns@qtronic.de, kehrer@iti.de

Abstract

We present and discuss the Modelica-based development environment currently used by Daimler to develop powertrain control software for passenger cars. Besides well calibrated vehicle models, the environment supports automotive standards such as A2L, MDF, CAN, and XCP to integrate control software and simulated vehicles on Windows PCs.

Keywords: automotive software development, software in the loop

1 Introduction

More and more automotive functions are implemented using software. Hence, there is an increasing demand to support the corresponding development process using virtual, i. e. simulation-based development environments.

development of control algorithms. This paper presents technology targeted toward the late stages in the development process, like tuning, validating and debugging the entire controller software in closed loop with simulated plant models. Virtualizing these later engineering tasks requires plant models with increasingly higher quality (physical effects modeled and quality of calibration) and near-production controller software (percentage of the controller software included, parameterization using production parameter sets and adaptation of the software to the plant) to be coupled.

A tool-chain supporting such coupling should

- be easy to set up and use by automotive developers who are usually not computer scientists
- support many of the engineering tasks usually performed with physical prototypes to allow for front-loading
- support short turn-around times, i. e. minimize the time between editing of control software and validation of the resulting behavior on system level to help find problems early
- provide built-in support for standards and de-facto standards used in automotive software development to allow cost-effective use of existing information sources
- support distributed development and exchange of work products between OEMs, suppliers, and engineering service providers. This requires e. g. measures to protect intellectual property.
- support reconfiguration of the development tool chain, since automotive development tools are frequently updated or replaced, e. g. due to emerging standards, new bus protocols or tool policy considerations.

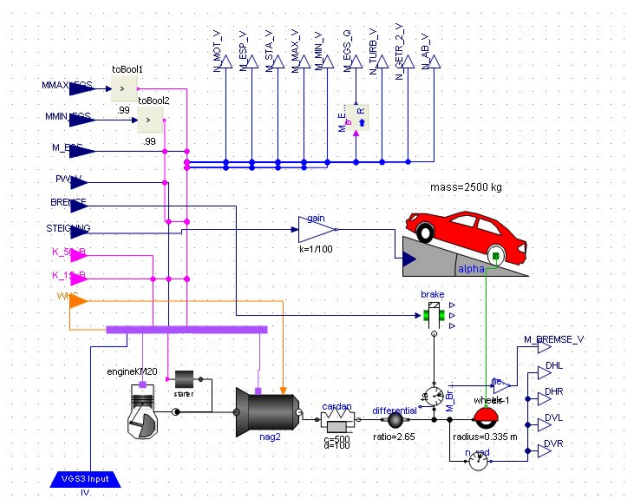


Figure 1: Vehicle model as used for SiL

Virtually coupling control strategies with plant models is standard technology today, mostly using common-place tools such as Matlab/Simulink for pre-

In this paper, we present the simulation-based development environment used by Daimler to develop the powertrain control software for Mercedes passenger cars. The tool chain presented here addresses the

above demands. It is based on vehicle models implemented using Modelica and processed using SimulationX as a tool for the design and analysis of complex systems, the FMU standard for model exchange, MATLAB/Simulink and TargetLink as a tool for model based development of automotive controllers and Silver as a tool for virtual integration of control software, application data and the simulated vehicle.

The paper is structured as follows: In the next section, we describe why and how Modelica is used here to create vehicle models. Section 3 describes how such a vehicle model is then coupled with control software and what else is needed to get automotive control software running in closed loop on a PC. Section 4 describes how such a SiL setup is used to support automotive software development, and section 5 describes costs and benefits of setting up a SiL.

2 Vehicle models

Daimler started around 2004 to use Modelica for building vehicle models used for test and development of powertrain control software via software in the loop (SiL). For example, the members of the 7G-Tronic transmission family have been developed this way [1]. Ongoing projects developed within this Modelica-based framework include dual-clutch transmissions by Mercedes [2] and AMG [3], and hybrid drivetrains. Basic requirement of a plant model in a SiL-environment for automatic gearboxes is the accurate calculation of the gear shifting. In order to achieve this goal, detailed model representation of gearbox kinematics, clutch mechanics and hydraulic control is essential. Therefore special Modelica libraries have been developed over the years to support transmission development.

For the development of customer specific libraries SimulationX offers a wealth of options such as the dedicated TypeDesigner that simplifies graphical and textual modeling compared to traditional forms.

Based on these libraries, a well calibrated vehicle model for a new transmission project can be setup within just a few weeks. This short development is partly credited to good properties of the Modelica language, which provides outstanding support for the reuse of component models, mainly by providing powerful means to parametrize models and built-in support for acausal modeling. Latter feature offers the model developer great possibilities to calibrate and validate his model by using measurements either from car or from test rig since no model modifica-

tions are necessary if the measured signal is a flow or potential quantity (e. g. torque as opposed to speed).

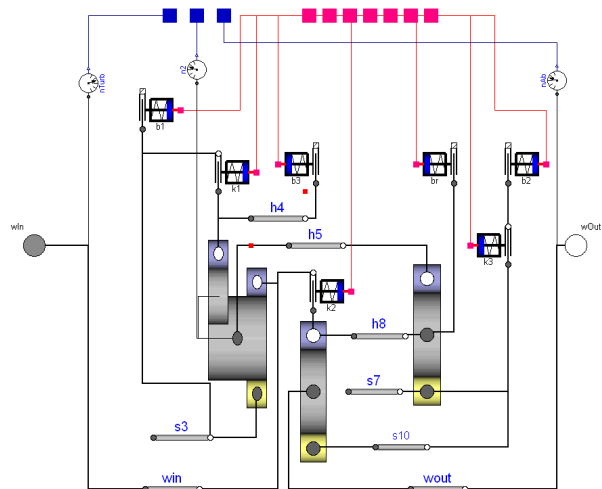


Figure 2: Gearset of a 7G-Tronic Transmission

Different capabilities for implementing measured data in SimulationX and validating the Modelica models against these data without the necessity of using another tool in combination with further options like the VariantsWizard help to increase the efficiency of model calibration. With special regard to the needs of powertrain modeling ITI provides different analyzing methods, e. g. the linear system analysis or the steady state simulation.

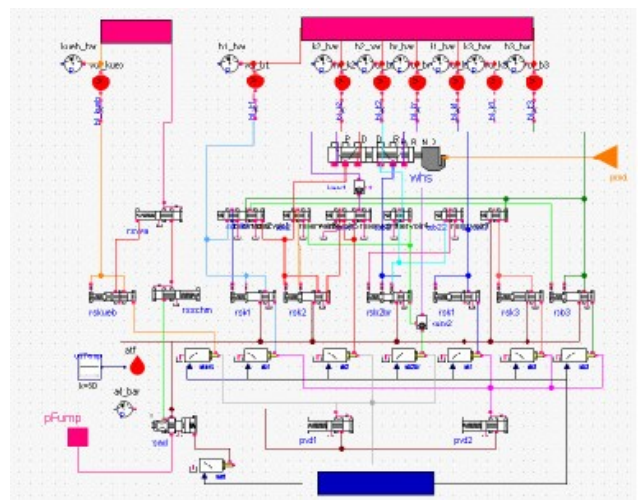


Figure 3: Transmission hydraulics

Figures 1, 2 and 3 show typical Modelica models used in series development projects.

Daimler uses Dymola and also SimulationX [4] to edit and process Modelica models. Since Modelica version 3.1 there is full compatibility of the plant models both in Dymola 7.4 as well as in Simula-

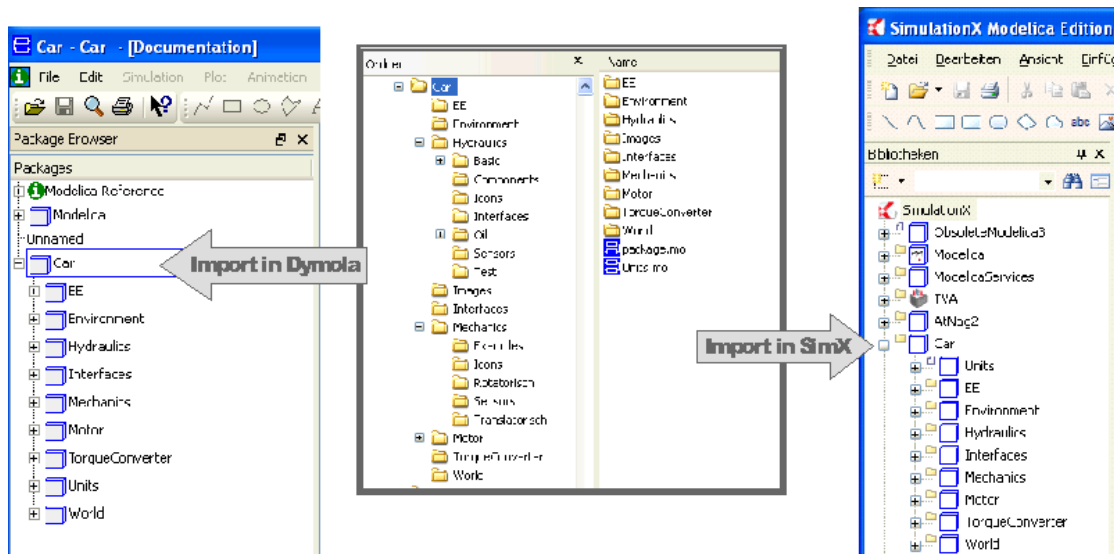


Figure 4: Modelica library Car in SimulationX and Dymola

tionX 3.4. Models and libraries are stored on hard disk as .mo files. Both tools are able to read these files with no specific modification, i. e. they use exactly the same files for displaying exactly the same structure. Figure 4 shows a screenshot of the directory structure and the integration in every tool.

This proves that one design goal of Modelica and the Modelica Standard Library (MSL) has been reached now, namely to provide a tool-vendor independent representation format for simulation models. There are however still a few issues to be solved to fully reach vendor independence of the MSL:

- The definition of tables in Modelica Standard Library is based on external functions. The implementation of these functions is not part of the library itself and has to be done by tool vendors. In consequence of missing specification the different implementations are not completely compatible.
- With the exclusive usage of external functions it is difficult to adapt the implementation on the requirements of the underlying tool. The substitution of external functions by external objects would improve the implementation capabilities.
- For users of a Modelica tool it is difficult to decide whether a used construct is compatible to Modelica language specification or not (e. g. classDirectory function). All tool dependent extensions of Modelica language should be marked as vendor specific similar to existing vendor specific annotations.

- Modelica libraries often use different version of annotations for graphical objects or attributes which are invalid in the particular context (e.g. fillColor for lines). While several tools ignore such annotations other programs generate error messages, which can be a little bit confusing for users and developers. For that reason a stronger validation of annotations would be preferable.

To create a Software in the Loop setup, the Modelica model is then exported. In previous years, the C code generated by either Dymola or SimulationX from a given Modelica model has been wrapped and compiled for execution by one of the SiL tools described in Section 3. For export, special wrapper code had to be developed for each simulation tool, and even for each version of such a tool, which was time consuming and error prone. Daimler started recently to use the FMI [8] developed within the Modelisar project as an export format for Modelica models. This standard is supported by the latest versions of SimulationX, Dymola, and Silver. This removes the need to maintain version and vendor specific wrapper code, which further improves and speeds up the SiL-based development process.

3 Getting automotive control software into the loop

Daimler uses Silver [5] and its in-house predecessor Backbone to virtually integrate vehicle models and control software on Windows PCs. Tools such as Silver or Backbone are mainly needed to support vari

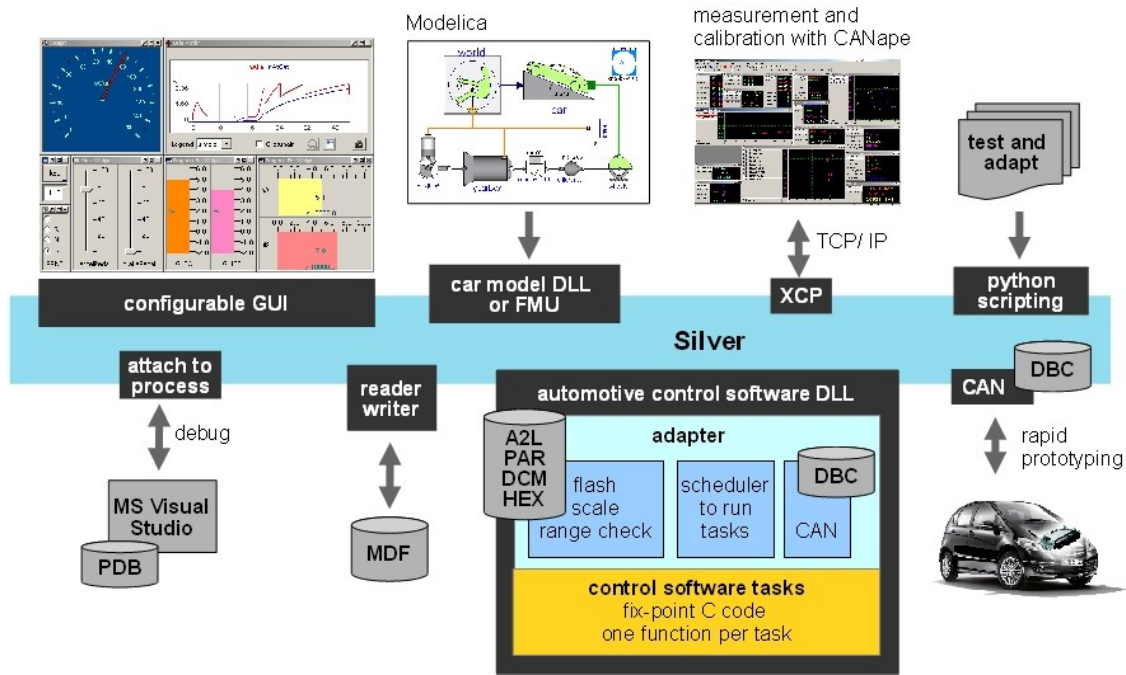


Figure 5: SiL environment and its interfaces to automotive standards

ous standards and quasi-standards used for automotive software development. Developers are familiar with these standards and know how to use them. Data is available in these formats already as part of the existing tool chain and reuse is virtually free of cost. Furthermore, using these data sources in the virtual development process allows early validation of these data sources. A virtual development environment should therefore mimic, emulate, or else how support these standards. A few examples of how the SiL tool supports automotive standards is shown in Fig. 5.

Developers typically use tools such as CANape (Vector) or INCA (ETAS) to measure signals and calibrate (fine-tune) parameters of the control software in the running car or on a test rig using standard protocols such as CCP or XCP. The SiL environment implements this protocol. Seen from a measurement tool such as CANape, a SiL simulation behaves just like a real car. Developers can therefore attach his favorite measurement tool to the SiL to measure and calibrate using the same measurement masks, data sources and procedures they are using in a real car. Likewise, automotive developers use MDF files to store measurements. The SiL can load and save this file format. A measured MDF file can e. g. be used to drive a SiL simulation.

Another example is A2L. This is a database format used to store key information about variables and (tunable) parameters of automotive control software. A2L contains e. g. the address of variables in the

ECU, its physical unit, comment and scaling information that tells how to convert the raw integer value to a physical value. The SiL-environment reads A2L files and uses the information to automate many tasks, such as scaling of the integer variables of the control software to match the physical variables of the vehicle model.

The SiL-environment also knows how to read DBC files. These describe how the control software communicates with other controllers using the CAN protocol. The SiL-environment uses this e. g. to implement rapid prototyping: Load the control software and the DBC into the SiL tool on your laptop, connect the laptop to car using a CAN card, and switch the ECU to 'remote control' mode. The control software running in the SiL tool controls then the corresponding system of the real car, e.g. an automatic transmission. The main advantage of such a setup is, that it saves time. Getting the control software running in a real ECU is typically much more time consuming than using a SiL tool or any other tool for rapid prototyping.

Finally, the SiL tool can process PAR and HEX files. These files may contain calibration data, i. e. values for all the tunable parameters of the control software. The SiL tool knows how to load these values into the control software running in the SiL, emulating thereby the 'flash' process of the real ECU. In effect, the SiL tool is actually not only running the control software, but the fine-tuned version of the software, which enables much more detailed investigation and testing of the control software's performance.

Having all these standards available in the SiL eases the task of actually getting automotive control software running on a PC, and doing useful things with the resulting setup. Control software is typically decomposed into a number of so-called tasks (i. e. functions implemented in C) that are run by an RTOS (real-time operating system) such as OSEK. Many tasks are periodically executed with a fixed rate, e. g. every 10 ms. To get such tasks running in SiL, the user has to build an adapter as shown in Fig. 5, i. e. a little C program that implements the Silver module API and emulates the RTOS by calling each task once at every (or every 2nd, 3rd, ...) SiL macro step. The SiL tool is shipped with the SBS (Silver Basis Software), i. e. C sources that make it easy to build such an adapter by adapting template adapter code. A cheap alternative to writing an adapter is to use the SiL tool's support for MATLAB/Simulink and Realtime Workshop (RTW). Automotive software is often developed by first creating a model of the controller using Simulink. The model is then used to automatically generate fix-point integer code, e. g. using tools like the Embedded Coder from MathWorks, TargetLink from dSPACE, or Ascet from ETAS (model-based development). The SiL tool contains support for exporting a Simulink model using RTW. The result will not use fix-point integer but floating point arithmetic, so it is Model-in-the-loop (MiL), as opposed to Software-in-the-loop (SiL). This is a fast push button solution for exporting a controller model to SiL, which does not require any hand coding, and is therefore attractive.

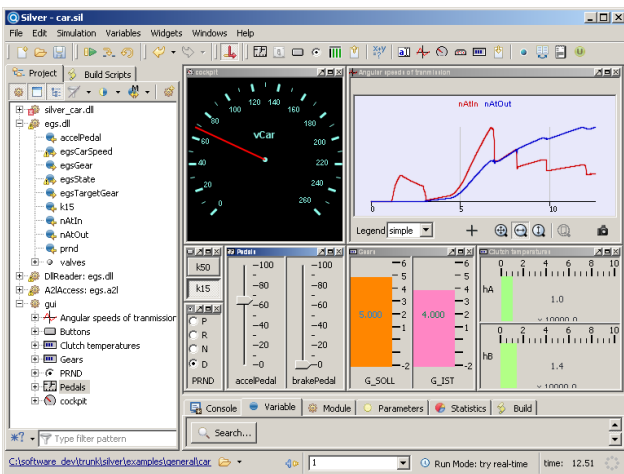


Figure 6: Software in the Loop (SiL) setup of transmission control software and vehicle model

4 Using the system model during automotive development

So far we have mainly described what is needed to get automotive control software running on a Windows PC, in a closed loop with the simulated vehicle. This section describes how such a SiL setup can then be used to support the development process. Supported activities include

- *Virtual integration:* Automotive control software for a single ECU typically consists of dozens of software modules, developed independently by a team of developers. Having a SiL helps to detect problems in the interplay of these modules early, long before an attempt is made to run all the module in a real car. For example, before releasing a new version of his module, a developer can quickly check on his PC whether the module works together with the modules of other developers. To do this, he only needs access to compiled modules (object files), not to the sources of other modules [2]. An additional benefit here is the isolation of developers from the changes of others when validating their modifications early on as his changes are only local to his own sources. Later integration efforts build on modifications already validated, albeit in isolation.
- *Debugging:* In contrast to the situation in a real car or on a HiL test rig, simulation can be halted in SiL. It is then possible to inspect all variables, or to change certain values to simulate a fault event. In conjunction with a debugger (such as Microsoft Visual Studio), it is even possible to set breakpoints or to single-step through the controller code, while staying in closed loop with the simulated car. The SiL tool can also be used to debug problems measured in a real car, if a measurement file (MDF) is available. In this case, simulation is driven by the measurement, and the SiL complements this measurement by computing the missing signals to provide a full picture needed to debug the problem.
- *Fault simulation:* Using a SiL, it is possible to create and explore scenarios that would be difficult or impossible to realize in a real car or on a test rig. For example, you can simulate strong wind [7] or inject arbitrary component faults into the simulation.
- *Comparing versions:* The SiL tool offers a function to compare the behavior of different

software versions by comparing all signals computed by these versions. This is e. g. useful when checking for equivalence after refactoring or clean up of modules.

- *Scripting*: A SiL simulation can be driven by a script, written e. g. in Python. This can be used to implement optimization procedures, for performing tests, or to trigger self-learning algorithms that adapt the control software to certain properties of the (simulated) car, e. g. to compensate aging of components.
- *Systematic testing*: In conjunction with the test case generator TestWeaver, the SiL tool allows the systematic testing of control software. TestWeaver generates thousands of test cases which are then executed by the SiL tool.
- *Virtual endurance testing*: calculation of load collectives for gearbox and drivetrain, e. g. to develop and test measures for safeguarding of the drivetrain components.
- *Application/Calibration*: of the control software on the PC.

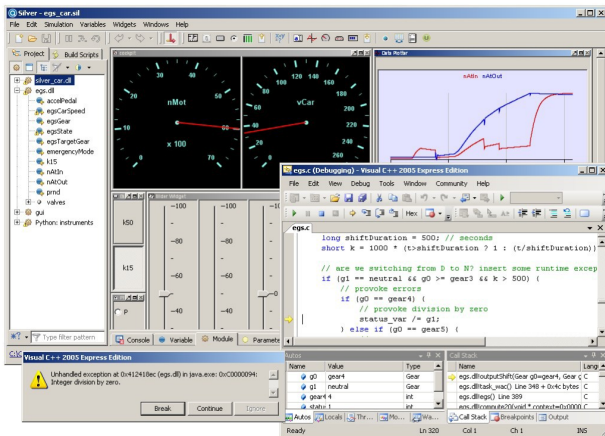


Figure 7: A debugger attached to Silver

A typical use case of the SiL tool is shown in Fig. 7. The test case generator TestWeaver [8] has found a scenario where the control software of a transmission performs a division by zero. This is clearly a bug. The user replays the recorded scenario, with Microsoft Visual Studio attached to the SiL tool. When the division by zero occurs, the debugger pops up as shown in the figure, showing the line in the controller source code that causes the exception.

5 Costs and benefits

Main cost factors of using the simulation-based tool chain for automotive software development are

- *development and maintenance of the simulation model*: Here is where modern modeling languages and tools such as Modelica and SimulationX help reduce costs by reuse of components and easy parameterization
- *continuous calibration efforts* to keep such a model up to date with the plant simulated: SimulationX allows continuous enhancements based on existing models and libraries by replacing components and models of varying complexity throughout all development phases. Reusing models including all interfaces necessary for calibration in combination with a wide range of tool options, e. g. VariantsWizard, COM-scripting or optimization tools, leads to an increasing efficiency in the workflow.
- *Building the adapter code for the controller software*: With the introduction of the Silver Basic Software package, this effort is significantly reduced.

Despite continuing cost-reduction efforts, these investments are still significant.

They are compensated by the benefits of such a Software in the Loop setup for developing control software, namely

- *extremely fast development cycles*: due to comfortable integration of software and vehicle components on the PC of the developer. This helps to detect problems early.
- *excellent debugging and test support*, e. g. with Microsoft Visual Studio Debugger or QTronic TestWeaver [1,2,3,6]. Found problems can be exactly reproduced as often as needed.
- *parallelize the development process*: A SiL configuration can easily be duplicated at low cost. This way, every member of a team can use its personal 'virtual' development environment 24 hours a day, without blocking rare resources like HiL test rigs, or physical prototypes.
- *sharing results without sharing IP*: All members of a team exchange working results by exchanging compiled modules (DLLs), not sources. This helps to protect intellectual property.
- *executing others contributions without their tools*: Our SiL runs modules (simulation models, control software) developed using

very different tools without accessing these tools. This greatly reduces the complexity of the SiL setups (no tool coupling).

6 Conclusion

We presented the tool chain used by Daimler for simulation-based development of transmission control software. The environment is based on Modelica, provides build-in support for automotive standards, imports vehicle models via the standard FMI and uses these models to perform closed-loop simulation of automotive control software. The virtual development environment created this way helps to shorten development cycles, eases test and debugging, helps to parallelize and hence to speed up development and provides a convenient platform for collaboration between Daimler's transmission development departments and its suppliers and engineering service providers.

Acknowledgments

Our work on the FMI [8] presented here has been funded by the Federal Ministry for Education and Science (BMBF) within the ITEA2 project MODELISAR (Förderkennzeichen 01IS08002).

References

- [1] A. Rink, E. Chrisofakis, M. Tatar: Automating Test of Control Software - Method for Automatic TestGeneration. ATZelektronik 6/2009 Volume 4, pp. 24-27.
- [2] H. Brückmann, J. Strenkert, U. Keller, B. Wiesner, A. Junghanns: Model-based Development of a Dual-Clutch Transmission using Rapid Prototyping and SiL. International VDI Congress Transmissions in Vehicles 2009, Friedrichshafen, Germany, 30.06.-01-07.2009
- [3] M. Hart, R. Schaich, T. Breitingner, M. Tatar: Automated test of the AMG Speedshift DCT control software 9th International CTI Symposium Innovative Automotive Transmissions, Berlin, 30.11. - 01.12.2010, Berlin, Germany.
- [4] SimulationX, <http://www.simulationx.com/>
- [5] Silver, <http://qtronic.de/en/silver.html>
- [6] A. Junghanns, J. Mauss, M. Tatar: TestWeaver - A Tool for Simulation-based Test of Mechatronic Designs. 6th International Modelica Conference, Bielefeld, March 3 - 4, 2008, pp. 341 - 348, 2008.
- [7] Hilf, Matheis, Mauss, Rauh: *Automated Simulation of Scenarios to Guide the Development of a Crosswind Stabilization Function*. 6th IFAC Symposium on Advances in Automotive Control, Munich, Germany, July 12 - 14, 2010.
- [8] FMI Specification 1.0, available for free from <http://www.functional-mockup-interface.org/>

Modelica Delft-Tyre Interface

Edo Drenth Magnus Gäfvert

Modelon AB

Ideon Science Park

Lund, Sweden

edo.drenth@modelon.se

magnus.gafvert@modelon.se

Abstract

The TNOⁱ Delft-Tyre is a renowned model for the pneumatic tire in the automotive industry based upon the famous Magic Formula first introduced by Bakker et.al. in the late eighties [1]. The name Magic Formula seems to appear first at the 1st Delft colloquium on tires four years later [2]. The Magic Formulae themselves have evolved greatly during the last two decades with contributions from a wide variety of companies and researchers.

The Magic Formula is widely used in the automotive (and gaming) industry because of its ease of use to represent the complicated tire characteristics.

TNO has marketed the Magic Formula tire model as Delft-Tyre and implemented the dynamic forces and moments computation routines, including the extension of SWIFT [3], in a variety of multibody simulation packages, like ADAMS and DADS and general purpose simulation software Simulinkⁱⁱ.

Modelon has in close cooperation with TNO by means of extensive benchmarks implemented the MF-Tyre/MF-Swift in Modelica. This paper presents the work conducted to implement the TNO tire models in Modelica which now is available as a commercial library in Dymolaⁱⁱⁱ.

Keywords: Delft-Tyre, semi-empirical tire model, Magic Formula, SWIFT-Tyre

1 Introduction

Modelica is gaining popularity as a modeling and simulation language. In order to further increase the number of possible applications by means of lowering the threshold to embark on the Modelica route, an interface to TNO's MF Delft-Tyre/MF Swift has been developed. Rather than following the object-oriented modeling path stipulated in [5], this interface creates a single wheel class that includes tire

characteristics and road-tire interface. One of the advantages using this solution, over the referenced one, is the well proven underlying code for force and moment computation of the tire characteristics. The tire model is semi-empirical and widely used in the automotive industry due to its ease of use and correlation with measured characteristics. Also, the necessary tire data is relatively easy and commercially available.

2 The models

2.1 Wheel model

The tire model used is TNO's MF-Tyre/MF-Swift. The MF-Tyre is solely based on the Magic Formula and can be used in models that are used for low frequency analysis (<8Hz), like exploring handling characteristics of a vehicle. The MF-Swift, that requires a separate license from TNO, can be used in models that are intended to be used in a higher frequency range up to 100Hz, and thus useful in for example ride and controller development studies. The MF-Swift tire model includes a rigid ring that models the tire belt and introduces additional rigid body modes to represent tire dynamics.

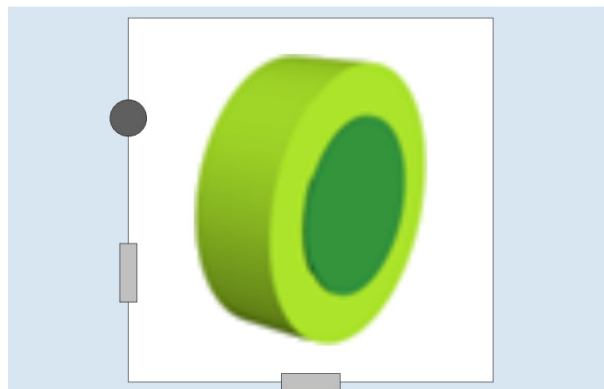


Figure 1 Wheel Model in Dymola (preferred)

The Modelica interface is firstly made available in Dymola and other simulation environments are in progress. In Dymola a wheel model (see Figure 1) is made available that incorporates the MF-Tyre/MF-Swift.

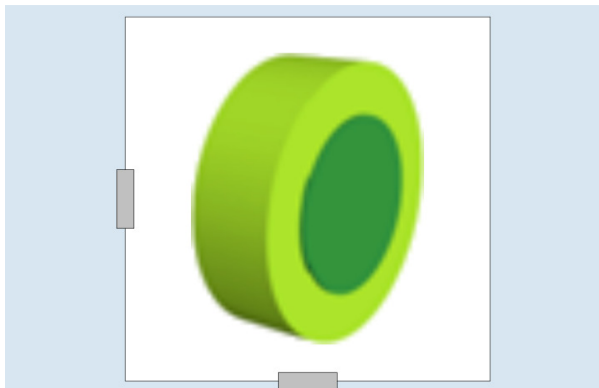


Figure 2 Wheel Model in Dymola

The preferred wheel model utilizes Dymola's capabilities of combining 1-DOF and 3-DOF multibody mechanics. The wheel is mounted with two connectors; a flange for the wheel spin degree of freedom and a normal connector for the wheel hub orientation. This is a numerical efficient method to include a power train in the complete vehicle models and compatible with the wheel implementation in Dymola Vehicle Dynamics Library, although not required for the Dymola Delft-Tyre interface.

TNO's Application Programming Interface, API, does allow for different types of interface and the straight forward interface with only one connector is also made available (see Figure 2). In this particular case the reference frame spins around the wheel spin axis.

2.2 Moving road

The API of MF-Tyre/MF-Swift allows, among different standard supplied types of road, moving roads. Moving roads are used to simulate for example a four-poster rig where the tire 'road' interface is moving vertically. Therefore a second connector is introduced in the wheel model that connects the wheel to the moving road and only visible when moving road is selected in the parameter dialogue.

3 Benchmark

3.1 Introduction

TNO has developed MF-Tyre/MF-Swift interfaces to three different simulation programs themselves. The

results of these three interfaces have been thoroughly tested with help of simple models and smart load cases to exercise all functionality and features. The entire virtual test program is specified in [7].

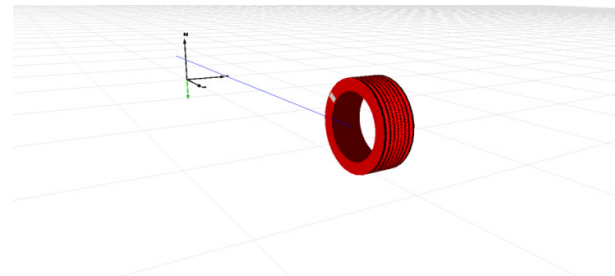


Figure 3 Single Wheel Model

In order to be allowed to release the MF-Tyre/MF-Swift interface for Modelica, sanctioned by TNO Automotive, models had to be made and run in Dymola according to the referenced specification.

The single wheel model (see Figure 3) is run many different ways in order to explore and verify the MF-Tyre/MF-Swift robustness and accuracy. For example, the tire model can be run in a steady-state mode, such that the relaxation length is omitted. In case of a motor cycle tire (with motor cycle tire data) the wheel model is run with significant camber velocity as well as spin velocity resulting in large camber to road angles.

Also, a simple vehicle model (see Figure 4) is simulated to extend the virtual tire tests. Modeling the ply-steer and especially conicity, the tires will have to be identified to be left- or right hand mounted.

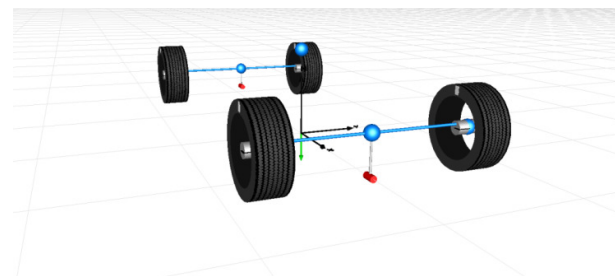


Figure 4 Simple Vehicle Model

The vehicle model will also be used on a four poster rig to verify the moving road interface. This four poster rig at the same time also verifies that the implemented tire model gives adequate results at stand-still.

3.2 Execution

The created models discussed in the chapter above had to be executed through the entire virtual extensive test program specified in [7]. The simulation results for all the different test runs will have to be

forwarded to TNO in a readable format. TNO will compare all results with their base line results with help of an automated process. The feedback by means of many graphical plots is sent back to the issuer of the simulation results. Never is the base line disclosed in numerical readable format. Once the simulation results match (some exceptions allowed, see chapter below) with the base line results, the interface is deemed to be implemented correctly. Experiences have shown this is a tedious process.

3.3 Results

During the course of executing the benchmarks and discussing and looping the results with TNO it became apparent that the specification in some instances was not concise enough and different interpretations could be made. Through discussion consensus was reached in all cases. In some cases the Dymola models had to be adjusted in other cases the specification had to be updated in order to aid future benchmarks with other simulation programs.

For instance the benchmark specification expected so called ‘cut-forces’ and ‘cut-torques’ between two bodies, where this has neither standard support in Modelica nor in Dymola. Specialized multibody simulation programs may have functionality to sum all forces and torques between two bodies, which becomes equivalent to cut-forces and cut-torques between two bodies. Cut-force and cut-torque sensors had to be created and inserted in the models at the right places.

Another discrepancy was found in one of the tests where the simple vehicle model had to perform a brake action on a split friction road surface. The benchmark specified a brake torque profile as function of time, but it became apparent that a special filtering function was used in the base line models to accommodate brake torques (see chapter below for detailed discussion).

4 Brake model

4.1 Introduction

As indicated in the chapter above, a discrepancy was found between the base line vehicle model and the Dymola vehicle model. The brake torque capacity was specified as function of time and an example of the front wheel brake torque capacity is depicted in Figure 5. However, the actual brake torque is basically limited by the actual surface friction between tire and road and the actual vertical tire force. Hence,

the actual resulting brake torque may be lower than the brake torque capacity, which is determined by the clamp forces, disc and pad friction and effective disc radius (in case of a disc brake).

4.2 Actual brake torque

Mathematically the brake torque is a result of the brake torque capacity and the external load on the wheel by means of longitudinal tire forces. The longitudinal forces on the tire are dependent on the load case as vertical tire force, actual surface friction and kinetic energy of the vehicle that the tire supports.

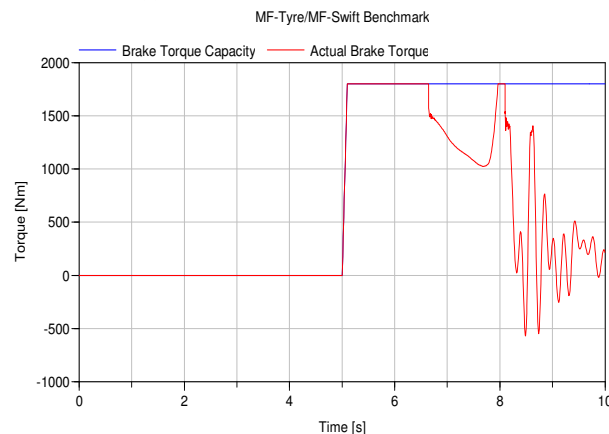


Figure 5 Brake torque capacity (blue) and actual brake torque (red) on front left wheel

The direction of the brake torque is also dependent on the load case and in its simplest form the brake torque takes the form:

$$T_{br} = T_{cap} \text{sign}(\omega_{wheel})$$

In many simulation programs the sign of the wheel spin velocity will cause numerical difficulties. Hence smoother sign functions are used. For example:

$$T_{br} = T_{cap} \tanh(\omega_{wheel})$$

Of course such solution will distort the actual result. Modelica allows, as a standard feature, for accurate friction torque modeling as discussed in [8].

4.3 Results

Figure 5 through Figure 7 show some Dymola results of the actual benchmark with the split friction braking maneuver.

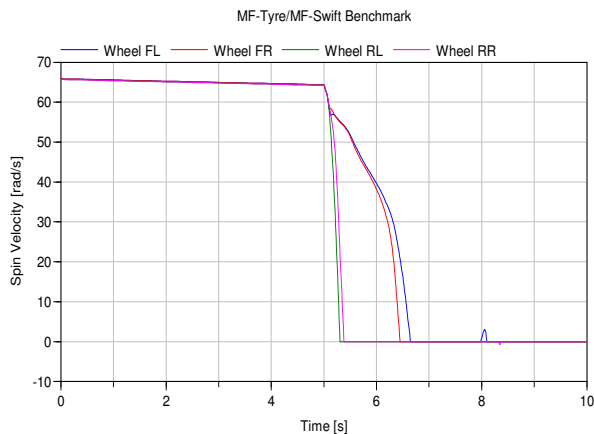


Figure 6 Wheel spin velocity traces

Figure 5 shows a trace of the brake torque capacity and the actual brake torque of the front left wheel. It can clearly be seen that the brake locks after about 6.5 seconds of simulation time, because the actual brake torque is lower than the brake torque capacity. At around 8 seconds of simulation time the brake unlocks a short while because the brake torque capacity and actual brake torque are equal.

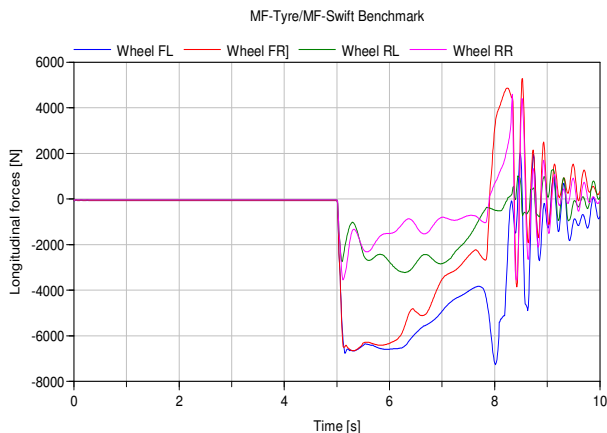


Figure 7 Traces of longitudinal tire forces

These results are also supported in the wheel spin velocity traces of Figure 6. The wheel spin velocity of the front left wheel becomes zero indeed after approximately 6.5 seconds of simulation time. Also, at around 8 seconds of simulation time the wheel spins up a split second.

For sake of reference the tire longitudinal force traces are shown in Figure 7. The vehicle will actually spin due to the yaw moment disturbance as a result of the split surface friction and the brake forces may switch sign.

The shown results deviate from the base line simulations run in non-Modelica based other simulation software as soon as one of the braked wheels locked, but were deemed correct due to modeling differences.

5 Conclusions

The extensive benchmark conducted in order to approve the MF-Tyre/MF-Swift Modelica interface has proven Modelica based simulation software Dymola to be a great tool to perform vehicle dynamics analyses. Yet with access to the renowned Magic Formula based tire model as a commercial library. This interface will leverage Modelica as simulation language for vehicle dynamics studies.

6 Outlook

Other Modelica simulation software solutions are in progress to adopt this interface.

References

- [1] E.Bakker, L. Nyborg, H.B.Pacejka, *Tyre Modelling for Use in Vehicle Dynamics Studies*; SAE 870421, 1987.
- [2] H.B. Pacejka, E.Bakker, *The Magic Formula Tyre Model*, 1st International Colloquium on Tyre Models for Vehicle Dynamics Analysis, Delft, The Netherlands, 1991
- [3] J.J.M. van Oosten, H.B. Pacejka, *SWIFT-Tyre: An accurate tyre model for ride and handling studies also at higher frequencies and short road wavelengths*, proceedings International ADAMS User Conference, Orlando, USA, 2000
- [4] H.B.Pacejka, *Tire and Vehcile Dynamics*, Butterworth-Heinemann, Oxford, 2002
- [5] M. Andres, D. Zimmer, F.E. Cellier, *Object-Oriented Decomposition of Tire Characteristics based on semi-empirical Models*, proceedings of 7th International Modelica Conference, Como, Italy, 2009
- [6] *MF-Tyre/MF-Swift 6.1.2 Help Manual*, TNO Automotive, The Netherlands, 2010
- [7] *MF-Tyre/MF-Swift Tyre Benchmarks, revision 1.3*, TNO Automotive, The Netherlands, 2010
- [8] M. Otter, H.Elmqvist, *Modeling and Real-time Simulation of an Automatic Gearbox using Modelica*, 9th European Simulation Symposium, Passau, Germany, 1997

ⁱ TNO - Netherlands Organisation for Applied Scientific Research

ⁱⁱ ADAMS, DADS and Simulink are trademarks of their respective owners

ⁱⁱⁱ Dymola is a Dassault Systèmes Modelica based simulation software

Multi-Domain Vehicle Dynamics Simulation in Dymola

Andreas Deuring^a Johannes Gerl^a Dr. Harald Wilhelm^b
^a Modelon GmbH München
^b Audi AG Ingolstadt

andreas.deuring@modelon.com johannes.gerl@modelon.com
 harald.wilhelm@audi.de

1. Abstract

In future cars, battery electric and hybrid electric drives will increasingly appear. Subsystems like e.g. the steering system and the braking system will accordingly be based on electric power supply. This leads to new challenges as well as opportunities also in the field of vehicle dynamics and an increased need of multidomain simulation concepts that combine multibody-based vehicle dynamics models and models of the electric and control systems. This paper includes a simulation study of the Audi sports car e-tron with electric power steering system using the Vehicle Dynamics Library from Modelon AB Sweden to model chassis and suspensions and the Modelica Standard Library to model the electric power steering system. The steering system controller unit was modeled alternatively in the Modelica Standard Library and in Matlab Simulink. Dymola and Matlab Simulink have alternatively been used as simulation environments whereas a special focus was put on different ways to integrate these tools according to standard development processes in the automotive industry. Additionally, extensive validation work was invested to compare vehicle dynamics results generated with ADAMS/Car and the Vehicle Dynamics Library.

2. Introduction

As hybrid and electric cars can store a higher amount of electric energy and dispose of higher voltage levels, it stands to reason to base subsystems like the steering system fully on electric power supply.

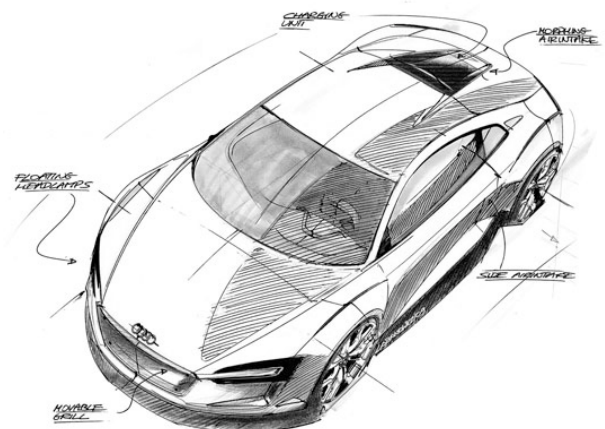


Fig. 1, Design Sketch of Electric Drive Sports Car Audi e-tron, Picture: www.audi.de

Moreover electric drive systems offer the opportunity to give controlled input to the wheel torque in order to optimize the handling and the safety of the vehicle. For example concepts which provide a combination of standard propulsion technology on one axle and electric drives on the other, offer certain potential regarding handling behaviour, however, require high attention to ensure safe driving in all conditions, e.g. during recuperation phases. As vehicle dynamics interfere with the dynamics of the electric systems and as an integrated control concept is required that includes vehicle dynamics and drive control systems, the usage of a multidomain simulation environment has obvious advantages compared to specialized tools with e.g. purely signal oriented or mechanical focus. In order to study the suitability of Dymola and the above named libraries, within this project a vehicle dynamics model of the Audi sports car e-tron was set-up and extensively verified and optimised towards an existing ADAMS/Car model. As an example for various electric systems, the electric steering sys-

tem was added to the vehicle dynamics model. Finally, the controller was modelled both in Dymola and Matlab Simulink. The entire system model was simulated within Dymola (importing the controller model via Functional Mock-up Interface [2]) and Simulink (importing the Modelica based models via the standard Dymola – Simulink Interface).

3. The Vehicle Dynamics Model Used

The model of the vehicle dynamics in terms of the mechanical system was carried out using the Vehicle Dynamics Library (VDL) [1]. It contains fully detailed multibody models of the double wishbone front and rear suspensions of the car, whereas the single suspension links are interconnected with nonlinear bushing elements. In Fig. 2 the Dymola Model of the front suspension linkage subsystem is shown in detail. The models are based on VDL standard templates.

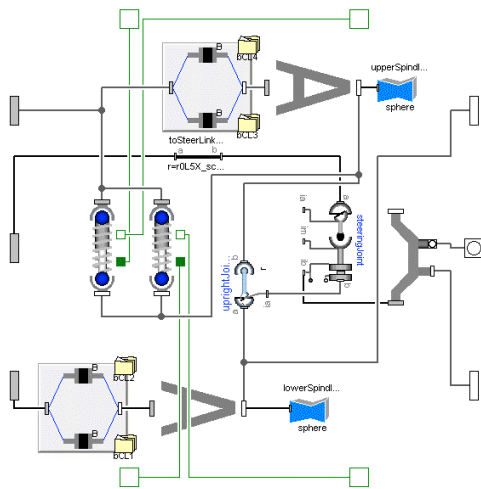


Fig. 2, VDL subsystem model of the double wishbone right linkage in Dymola and Vehicle Dynamics Lib

Fig. 3 gives a showcase overview of the full vehicle as it is graphically displayed in the animation tool of Dymola. The focus of the modelling work lied on the resolved rear and front suspensions. The car body and the subframes are modelled as rigid parts having six degrees of freedom each. The structure of the VDL-model was based on an already existing ADAMS/Car model with comparable complexity. The assembly of the VDL subsystems with the relevant multibody data like masses,

inertias, geometry points, elasticities, damping, etc. was transferred and adopted from this ADMAS/Car model.

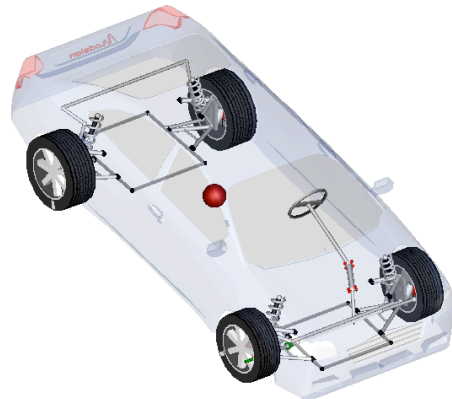


Fig. 3, Graphical animation of the full vehicle model in Dymola with focus on the resolved suspensions

However, the ADAMS model was not realised in any detail, as this was not in focus of the project. Differences in the simulation results aroused from certain elasticities of the suspension models that have not been taken into consideration in the VDL, due to the wish to work with standard templates.

4. Comparison of the VDL and the ADAMS/Car model

To compare and validate the VDL towards the ADAMS/Car model, experiments in the field of suspension kinematic and compliance analysis (K&C) have been set up as well as full-vehicle handling experiments. For the K&C tests, elasto-kinematic models have been created, optimised and then used in identical form in the full vehicle analysis. The results shown exemplarily in the following two figures contain a small cutout of the entire set of results that was achieved.

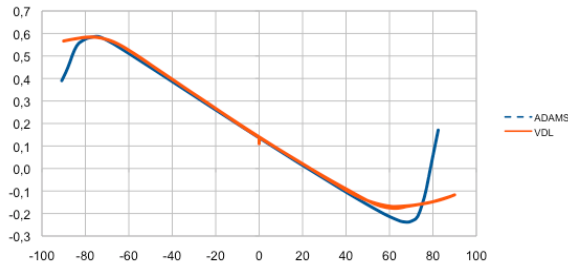


Fig. 4, Front Axle K&C-Test: Toe Angle (degree) vs. Wheel Travel (mm)

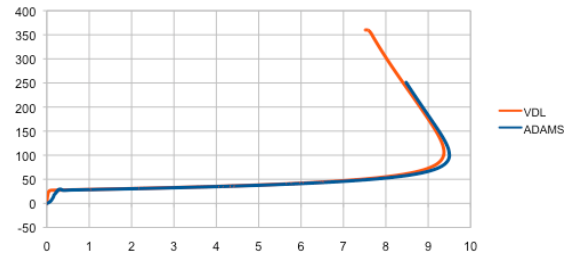


Fig. 6, Full Vehicle Steady-State Cornering: Steer Wheel Angle (degree) vs. Lat. Acceleration (m/s^2)

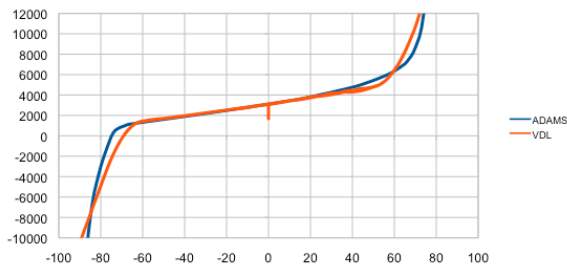


Fig. 5, Front Axle K&C-Test: Vertical Force (N) vs. Wheel Travel (mm)

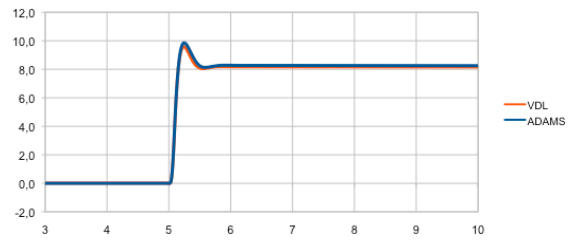


Fig. 7, Full Vehicle Step Steer: Yaw Rate (degree) vs. Time (s)

In summary the K&C results achieved the expected accordance between VDL and ADAMS or have explainable deviance due to differences in modelling. The quality of K&C accordance was from a certain point on not further optimized, as not being in the focus of the project. E.g. the differences in the extreme regions of Fig. 4 and Fig. 5 are due to a different modelling approach for the bump stop.

A typical set of entire vehicle handling experiments was carried out, too. Exemplarily for the comparison an abstract of results from the fundamental experiments *Step Steer Manoeuvre* and *Stationary Cornering* are given in Fig. 6 and Fig. 7.

As tire model Pacejka's magic formula was used in both codes.

In summary the full vehicle simulations are in good accordance. It could be shown that an industry standard vehicle dynamics model in ADAMS can be redone with reasonable effort and satisfying precision within a relatively short time in a multidomain simulation tool. Having available the model there, additional non-mechanical systems can be added easily.

For the following investigations no further comparisons to the ADAMS/Car model were considered.

5. Electric Power Steering and Controller Model

The Electric Power Steering (EPS) model was added to the vehicle dynamics model described above in Dymola, using elements exclusively from the Modelica Standard Library. A model of the steering controller was created in Dymola, too, and alternatively the controller was added to Dymola as a Simulink model that was exported with the Real-Time Workshop using the Functional Mock-up Interface (FMI) [2]. The FMI was defined by the Modelisar consortium with the intention that dynamic system models of different software systems can be exchanged and used together for simulation. The Functional Mock-up Unit

(FMU) essentially contains an Xml model description and a Dynamic Link Library (DLL).

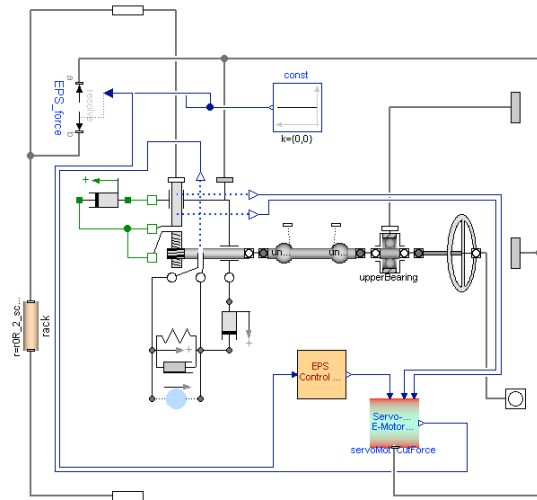


Fig. 8, Electric Power Steering and Controller Model in Dymola

The EPS model itself consists of mechanical multibody, electric and control blocks. The controller was simplified and treated as a black box of binary code, as this is the usual way that vehicle manufacturers receive the model code from their system suppliers. For this kind of pilot approach the controller was kept as simple as necessary and was designed just as a proportional gain. The output of the control unit is a drive signal for the voltage of the electric motor.

The impact of the power steering system on the dynamic vehicle behaviour was not the point of interest in this project and therefore not elaborated or tuned.

6. Interface Concepts and according Simulation Results

Dymola and Simulink models can be interfaced in multiple ways. For instance, the Simulink model of a controller can be imported to the Dymola model using the Functional Mock-up Interface Approach proposed by the Modelisar research project [2]. In this case Dymola serves as the solver for the entire system consisting of Modelica and Simulink subsets. Alternatively Dymola models can be exported to Simulink using e.g. the standard s-function interface of Dymola. In this case Simulink serves as the solver.

According to the scope of the simulation work and the particular development process of the user, there are motivations for both ways. For this project, the following variants have been applied.

	Vehicle Dynamics Model	EPS Actuator Model	Steering Controller Model	Simulation Tool, Solver
Variant 1	Dymola VDL	Dymola Electric Lib	Dymola Control Lib	Dymola
Variant 2	Dymola VDL	Dymola Electric Lib	Simulink	Dymola
Variant 3	Dymola VDL	Dymola Electric Lib	Simulink	Simulink

Table 1, Different Interface Approaches for the Simulation of a Vehicle with EPS-System

Due to the multidomain approach of the simulation concepts described above, in any variant multiphysical results can be studied, illustrated e.g. by the analysis of the EPS motor current in Fig. 9.

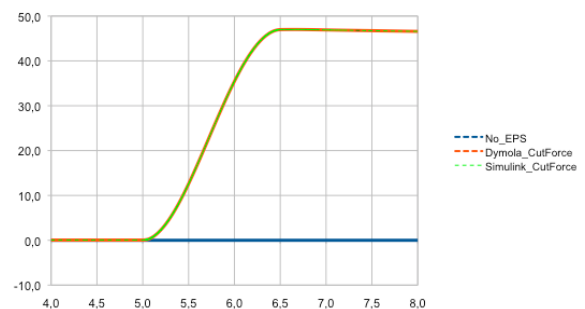


Fig. 9, Full Vehicle Step Steer: Steering Servo Motor Current (A) vs. Time (s)

Due to the different solver technologies and ways to derive equations from the system description, significant differences in the computational performance of the studied interface concepts occur for a **Step Steer Manoeuvre** (SSM) and **Steady State Cornering Manoeuvre** (SC). In all cases, however, the simulation results are practically identical.

	Simulation Tool, Solver	Simulation Time, SSM	Simulation Time, SC
Variant 1	Dymola	25,2 sec.	14,8 sec.
Variant 2	Dymola	25,2 sec.	14,8 sec.
Variant 3	Simulink	25,3 sec.	65,4 sec.

Table 2, Simulation Performance of Different Interface Approaches

References

- [1] Niklas Philipson, Johan Andreasson, Magnus Gäfvert, Andrew Woodruff: Heavy Vehicles Modeling with the Vehicle Dynamics Library, Modelica 2008, March 3rd – 4th, 2008
- [2] MODELISAR consortium. Functional Mock-up Interface for Model Exchange. V. 1.0. <http://www.modelisar.org>, 2010

7. Summary and Outlook

The work presented demonstrates that a resolved multibody model comparable to an industry standard ADAMS model can be created with reasonable effort in a multidomain simulation environment like Dymola using the Modelica approach and according specialised libraries. Extensive validation work was invested to ensure that both models lead to comparable results.

From there on it was demonstrated that entire mechatronic system simulation is easily possible in multidomain simulation tools, using vehicle dynamics, electric, additional mechanical and control models. Manifold ways to interface Dymola and Simulink support flexible approaches and tool strategies to simulate multiphysical mechatronic systems and match the particular needs of a user’s specific development process.

It was shown that the described approaches have the potential to cover the needs of the upcoming challenges of e-mobility for system design.

Additional concepts to interface multidomain simulation tools like Dymola with control simulation tools like Simulink are under development at different places and promise an even tighter integration of the required tools.

Modelling and Optimisation of Deviation in Hydro Power Production

Telemark University College

Dietmar Winkler Hege Marie Thoresen Ingvar Andreassen
Magamage Anushka Sampath Perera Behzad Rahimi Sharefi

Abstract

In 2009 a new price system for the produced power was introduced for Norwegian hydro power plants. Basically a power producer gets punished if his production deviates from the scheduled production. The power plant is paid for the actual production: if too much power is produced that the price for the excess power is low. Even worse if too little power is produced the power plant has to pay a fine.

If such deviation occurs it is very important to identify components/systems that are responsible in order to adjust the controller or replace the faulty equipment. This paper describes the first step in problem solving, by presenting the development of a model of a hydro power plant that shows differing power production. The modelling part was done in Modelica[®] using the HydroPlant Library¹ of Modelon AB. The model was parametrised using construction data and validated using data from test and operation runs.

Keywords: Modelica, Hydro Power Systems, HydroPlant Library, Test and Validation

1 Introduction

Several things affect the actual production in the power plant. The turbine governor controls guide vanes in the drum case, which in turn determines the power production. The most important input is the set-point. In addition it is possible to vary how sensitive the controller is wrt. changes in the frequency (also known as *droop control*). The droop is set by the national grid company, *Statnett* in case of Norway.

The produced power also varies with the water level in the reservoir. With a fixed guide vane opening the

produced power increases when the reservoir level increases and decreases when the level decreases. The reservoir level is also fed to the turbine controller as an input and the controller should in principle compensate varying levels automatically.

In general, the various measurements, other inputs, and transformed signals used as inputs to the turbine controller, can contain uncertainties which may lead to production deviation.

In the past it occurred that a power plant called “Sundsbar” which is located in Seljord, Norway has experienced a certain power production deviation. In order to investigate the reason for this deviation a reference model using the Modelica modelling language was developed and validated. With the help of this model ideally the reason for production deviation could be identified.

2 Main Components of a Hydro Power System

Hydropower facilities regarding if they store water for peak load period or not, and the way they store the water, can be classified into several categories:

Storage Regulation (Impoundment) This is the most common development of hydroelectric power plants in which a dam is used to store a large quantity of water in a big reservoir. Potential energy of the stored water then can be released in a controlled way.

Diversion Part of the river water is diverted into a canal or a tunnel and is passed through the power station and then might join the river again in the downstream. This development can use the natural difference in height of the river at the upstream

¹For more information on this library see: <https://modelica.org/libraries/HydroPlant>

and downstream locations and may not require a dam.

Run of River In the run-of-river type, a small dam with little impoundment of water is used. Short tunnels (called penstocks) direct water to the power station using the natural flow of the river. Capacity of generating electricity in a diversion or a run-of-river station is dependent on the amount of water flowing in the river.

Pump Storage In pump storage development water is pumped to a higher reservoir during periods of low energy demand. The water is then run down through the turbines to produce power to meet peak demands.

Hydropower stations may also be classified by the type of their loads into Base-Load and Peak-Load plants [1]. The main focus of this paper will be on the Storage Regulation (Impoundment) type of hydropower stations.

2.1 The Water Way

Figure 1 shows an example cross-section of an impoundment hydropower station. The water passage starts from the upstream reservoir and ends at the downstream pond/river. The difference of elevation between water surfaces in the reservoir and the downstream pond determines the total head (gross head) of the water in the system. Because of the energy loss in the water passages due to friction, the effective head of the water that can be exploited is less than the total head.

Different parts of the water passages in this type of power plant are described briefly.

Intake Intake is the inlet of the head race tunnel which is equipped with Trash Rack for preventing big solid objects from entering the tunnel and Gate Door for isolating the tunnel for maintenance.

Head Race Tunnel or Conduit Head race tunnel (conduit) connects the reservoir to the penstock near the power house. It can be equipped with sand traps for collecting sand and garbage that had passed through the trash rack in the intake. The tunnel can be as long as 45 km like in “Muela” power station in South Africa [2]. It is also possible that the tunnel have inlet branches from more than one reservoir.

Surge Tank/ Surge Shaft Surge tanks or surge shafts might be used in different parts of the water passage to prevent the “water hammer” effect. When the water flowing in tunnels or pipes is accelerated or decelerated, pressure surges are created in the waterway which their magnitude may be much larger than the nominal pressure in the waterway. This effect is known as “water hammer”. The strength of the pressure surges depends on the value of acceleration/deceleration and the length of the waterway. The waterway shall be built strong enough to withstand these pressure surges (often not an economical solution) or the surge magnitude shall be limited/ controlled somehow. One way for reducing the surge amplitudes is using surge shaft/ surge tank. Some other means of controlling the water hammer are:

- Limiting the gate or valve closure time
- Using pressure regulator valves / relief valves located near the turbine

Penstock A penstock (also called Pressure Shaft) is a pipe made of concrete, steel, fiberglass, or wood that is used to carry water from the supply sources to the turbine. This conveyance is usually from a canal or reservoir or from a tunnel. Penstocks may be equipped with shut-off/isolation valves. The control valve (e.g., guide vanes) at the turbine side regulates the water flow through the turbine.

Turbine Case The turbine case is the final component of the water passage before the guide vanes and the turbine runner. Here turbine cases for Francis turbine will be described shortly.

Guide Vanes Guide vanes are located between the turbine case and the runner of Francis turbines. These are movable vanes that are actuated by turbine governor to control the flow rate of water through the turbine and hence controlling the load of the turbine.

Turbine Runner Turbine runner for a Francis turbine is a reaction type turbine. Instead of using a water jet (like for impulse type turbines like Pelton turbines) a water flow is allowed to pass through the runner.

Draft Tube Draft tubes are the final components of the water passages of hydropower plants with Francis and Kaplan Turbines. Draft tubes carry away the water from turbine to the downstream channel or pond (the tail race).

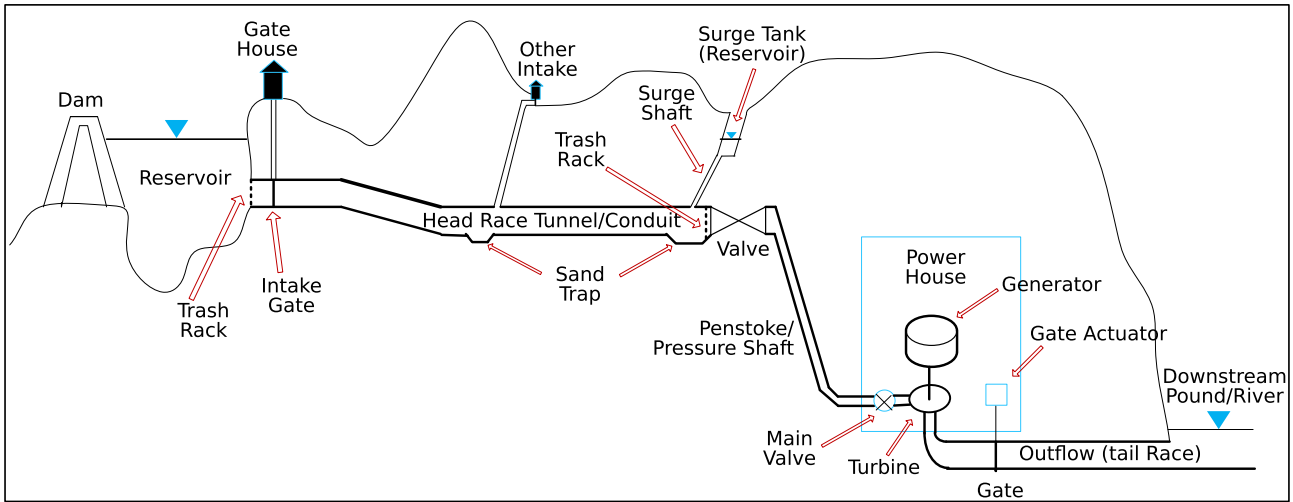


Figure 1: Example Schematic Diagram of an Impoundment Hydroelectric Station

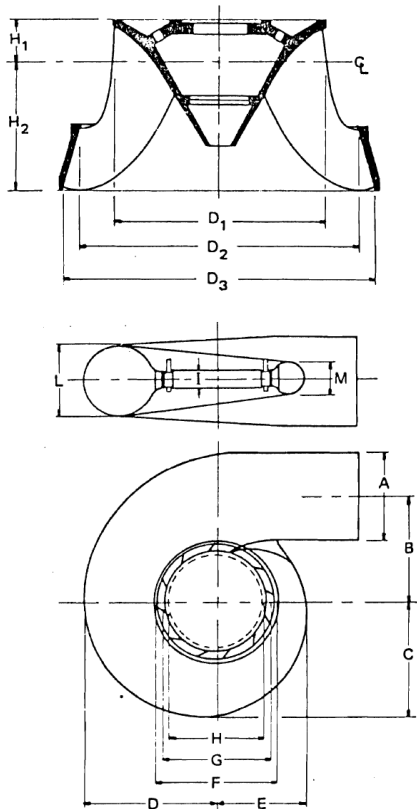


Figure 2: Francis Turbine Case with Dimensions [1]

Usually spiral cases are used for delivering water to the Francis turbine. Figure 2 shows a typical Francis turbine and its case. In general, following considerations shall be taken in design of the shape and dimensions of the case and runner:

1. The cross-sectional area of the spiral case must

decrease so that the turbine runner is supplied with water uniformly around its circumference.

2. Irrationality of the flow inside the case shall be maintained.

2.2 The Electrical System

It follows a very brief introduction into the generator theory.

The generator converts the mechanical energy from the turbine into electric energy. The basic principle behind a generator was discovered by Michael Faraday, that a voltage is induced in a conductor when it moves through a magnetic field. Faraday's law of electromagnetic induction is explained in equation (1):

$$emf = -N \frac{d\phi_B}{dt} \quad (1)$$

Where

emf = electromotive force [V]

ϕ_B = magnetic flux [Vs]

N = number of wires in the conductor

The Generator has two main parts: The rotor, which is the rotating part, and the stator, which is the stationary part. The rotor is delivering the magnetic field, and the copper coils in the stator get an induced voltage from the rotating magnetic field. There are two main types of generators, synchronous generators and asynchronous generators. The synchronous generator is the most used generator in bigger hydro power plants. Smaller hydro power plants may have asynchronous

generators, such as smaller hydro power plants which produce up to about 5 MW.

2.2.1 Synchronous generator

The synchronous generator has a DC electric field in the rotor. In reality, a reverted synchronous AC generator (with armature windings on its rotor) connected at the end of the synchronous generator shaft produces this DC current. A principal sketch of a two-pole, single-phase synchronous generator is shown in Fig. 3. It shows the field conductors in the rotor which makes the magnetic field, and the stator conductor which gets an induced voltage from the rotating magnetic field. When the rotor turns and the poles change place, the induced voltage in the stator is alternated. This makes the generator produce AC (Alternating Current) voltage from the DC current in the rotor. The terminal voltage of the generator can be controlled by the magnetising current in the rotor.

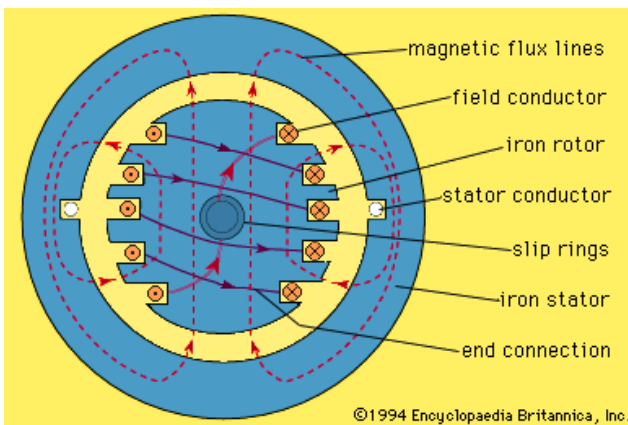


Figure 3: A 2-pole single phase synch. generator [3]

2.2.2 Power factor

The power factor is a very important subject when it comes to electric power. When the voltage and current are in phase (no lag) then the power factor is maximum, i.e., 1. This gives the optimal power output. The definition of power factor is:

$$PF = \frac{P}{S} = \cos \phi \quad (2)$$

Where

$$\begin{aligned} S &= \text{apparent power [VA]} \\ P &= \text{active power [W]} \end{aligned}$$

2.3 Control System

2.3.1 Turbine governor

The turbine governor's main task is to control the power output and the rotational speed of the turbine and also to smooth out differences between generated and consumed power at any grid load and prevailing conditions in the water conduit [4]. At the same time the governor also need to close down the admission at load rejections or when a need for an emergency stop rises. This has to be done in accordance with specified limits of rotational speed and pressure rises in the waterway.

Deviation between power generation and consumption in the grid will cause an acceleration or deceleration of the rotating masses of generating units. Acceleration happens in case when generation is more than consumption. The turbine governor then will cause a deceleration of the water flow. At the same time pressure in the penstock will increase.

In order to keep the rotational speed within specified limits at load rejections the admission – closing rate must be equal to or higher than a given value [4]. In the opposite way the closing rate of the admission have to be equal or lower than a certain value in order to keep the pressure rise in the conduit within specified limits. The turbine governor acts in two modes: speed control and load control. Speed control mode takes place when the generator is isolated from the grid (MCB is open). In this mode the governor regulates the speed of the turbine-generator with the speed set point. Load control mode takes place when the MCB is closed. In this mode the governor regulates the generated power with the load set point and through a mechanism called “droop” which is described in section 2.3.2. The governor output signal in a Francis turbine power generation unit is applied to the guide vane servomechanism and hence the governor controls the unit through the guide vane position.

2.3.2 Speed Droop Control

In case of frequency increase (decrease) in the grid, each power generation unit reduces (adds) a fix percentage of its total rating output power multiplied by the amount of the change in the grid frequency' from (to) its output power. The amount of this power can be calculated from equation (3):

$$S = \frac{\Delta f / f_N}{\Delta P / P_N} \cdot 100\% \quad (3)$$

Where

$$\begin{aligned} S &= \text{permanent speed droop [\%]} \\ f_N &= \text{nominal frequency [Hz]} \\ P_N &= \text{nominal power [MW]} \end{aligned}$$

Where the permanent speed droop in equation (3) is a percentage number which is decided by the grid administration (e.g., Statnett in Norway). For example in Norway for a stable operation of the electrical grid the permanent speed droop is currently set to 10% .

2.3.3 Power/Frequency Control

In every electrical system the power needs to be produced when it is consumed. It is not possible to store electrical energy. Energy has to be stored in the form of reservoirs for larger power systems, and as chemical energy (batteries) for small power systems. This means that the production system must be sufficiently flexible to both changes in consumption and the outcome of the production, and that the transfer can be handled instantaneously, preferably without consumers noticing it. For example, the national grids in Norway, Sweden, Finland, and at Sjælland in Denmark are all connected to one coordinated synchronous grid. This means that events in one of the sub-grids can affect the other grids in the other countries.

The frequency is a measure of how fast the machines in the system rotate. If it becomes an increase in load (as with any other rotating machines) the frequency (speed) will decrease, and at load rejection the frequency will rise. The controlling devices will automatically perform a primary control so that it again is a balance between production and consumption. How much the speed decreases are influenced both by the total torque, and by how quickly the primary control is done.

At frequencies below 50 Hz, the total load will get higher than the desired production and at frequencies above 50 Hz, total load will become lower. In practice, the load varies continuously. Consequently, the controlling devices continuously need to perform the frequency control.

The power/frequency control is normally exacted in two stages.

1. Primary control or primary frequency control is simply the application of the speed droop control as mentioned in section 2.3.2. This kind of control is applied automatically and is built into all

turbine governors. This means that when the frequency deviates from the optimal 50 Hz the turbine governor will increase or reduce the guide vane opening according to the droop control settings.

2. After the primary control has settled we will still have a constant frequency deviation. This is when the secondary control is used to compensate the deviation with the help of the *Load Frequency Control* (LFC). The LFC will simply raise or lower the set-point so that frequency is corrected again.

LFC is normally used in combination with *Automatic Generation Control* (AGC) where different generation regions are taken into account in order to balance the power production [5]. However internationally there exist different interpretations and implementation of the Automatic Generation Control.

First the primary frequency control is applied.

3 Complete Dynamic Model

This section discusses the basic hydraulic theory related to hydropower plants, hydro power modelling in the Modelica HydroPlant Library (HPL), analytical models for hydropower plant's hydro dynamics and analysis. The total system is divided into several subsystems, namely reservoir, conduit, surge tank, penstock, turbine, generator, and grid. The HydroPlant Library uses digitised turbine characteristics (in practise turbine characteristic is given as a chart so called "Hill Charts" by the manufactures) and this may lead to some uncertain results (because some extrapolations and interpolation needed among data points). So emphasis is put on having a good analytical model for turbine. An analytical model for a Francis turbine is proposed in [6].

In the HydroPlant Library some units (e.g., conduit, penstock) are divided into sub volumes also called control volumes (CV) and each sub volume is characterised by temperature and pressure (so called the state of a control volume). Two Ordinary Differential Equations (ODEs) are derived from the conservation equations (i.e., mass, energy). Mass flow rate between two adjacent control volumes is governed by a third ODE which is derived using the momentum conservation. The main assumption is that state is uniformly distributed throughout the CV. This is the so-called "Lumped Parameter" assumption.

Two dynamic equations will be derived for temperature and pressure. But however more concern is given into deriving equation in Laplace or frequency domain which will help to study the dynamics of the systems. A detailed discussion is given in [6]. When a hydropower plant is modelled (in Modelica), local resistances (e.g., trash rack losses, bend losses) are considered to be *minor* pressure losses while the *major* losses are due to wall friction. So in the Modelica model those minors losses are neglected, only major wall frictional losses considered (HydroPlant Library blocks take care of this).

4 The Sundsbarm Hydro Power Plant

In this section only some parts of Sundsbarm Hydro Power Plant are explained.

Francis Turbine governor The turbine governor at Sundsbarm is a TC 200 digital turbine governor from Kværner. It is a PID controller, and it has inputs for frequency reference (f_0) and frequency measured out from the generator (f), as well as inputs for load reference (P_0) and power measurement after the generator (P).

The set point for frequency and the permanent speed droop are controlled and set by the operator or a overall control system.

The functions inside the governor at Sundsbarm and the function of the governor used in the HydroPlant Library are not identical. In order to obtain as similar control of the power plant inside Modelica and the real process at Sundsbarm these functions need to equal each other. In order to get this similar to each other we have looked into the block drawings and made a simplified block drawing of the functions inside the TC 200 governor at Sundsbarm. We then compared this with the block drawing from the HydroPlant Library and we obtained the controller in Fig. 4.

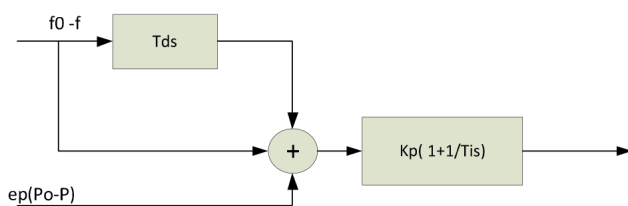


Figure 4: Simplified block-drawing from the turbine governor at Sundsbarm

This structure can be reorganised to the controller shown in Fig. 5.

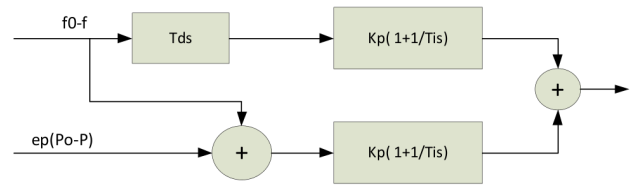


Figure 5: An equivalent controller block diagram to the one at Sundsbarm

This gives two governors:

- One with PI characteristic
- One with PD characteristic

Turbine The turbine placed at Sundsbarm is a vertical Francis Turbine with a performance of 104.4 MW, and is constructed for a nominal head (H_{nom}) of 460m. The efficiency and volume flow of the turbine is calculated in the Modelica model using a look-up table.

Generator and Main Circuit Breaker In Sundsbarm there is one generator delivered by “National Industri” Drammen, Norway. The nominal performance is 118MVA and the nominal frequency is 50Hz. The number of generator poles is 12. Inertia momentum of the generator is given by Alstom and is equal to 850,000 Kg.m².

Reservoir The reservoir type for the Sundsbarm plant is an impoundment dam. This means that the water source has a water storage that makes it possible to store energy in the reservoir. The lake *Sundsbarmsvatn* is used as a reservoir. Other lakes are connected to *Sundsbarmsvatn*, to lead more water through the power plant. The other lakes are not modelled, because they are not connected directly to the conduit channel or to the penstock.

The Sundsbarm lake is approximately 12.5km long and 0.75km wide. These approximate sizes of the reservoir are used in the model, because normal operation conditions are of interest, not the level of water varying over a long period in *Sundsbarmsvatn*.

Conduit channel The conduit channel consists of the intake at the reservoir, *Sundsbarmsvatn*, a trash rack, and an intake gate. Just before the penstock

there is another trash rack, a surge shaft and an emergency valve. There is an additional small intake called *Finndalsåi* that is connected directly to the conduit channel but was not included in the model.

Penstock The penstock for Sundsbarm hydro power plant consists of a steel pipe inside a tunnel. The penstock is 600 meters long and is tilted 45° . The start of the penstock is at 541.5 m above sea level and the end of the penstock is at 112.5 m above sea level. Diameter of the penstock is about 3 meters.

Surge shaft The surge shaft for Sundsbarm hydro power plant is a pipe or tunnel with a length L of about 138 m which is tilted by 67.5° .

Outflow tunnel The outflow tunnel consists of a rough tunnel that goes from the draft tube at the turbine and to the output reservoir. The outlet tunnel has its lowest elevation closest to the draft tube at 107.5 m . At the end of the tunnel the elevation is 4.5 m higher, at 112 m above sea level.

Reservoir at outlet The reservoir at the outlet is the river in *Seljord*. The reservoir model used here is also the `Fixed_HT` model together with a model of the reservoir. The Fixed HT model has an infinite volume with prescribed water height and temperature. The level at the outlet reservoir is 123 m above sea level.

4.1 Modelica Model of Sundsbarm Power Plant

Putting all the different parts together using the components from the `HydroPlant` Library and filling in their respective parameters we gain the complete dynamic model as shown in Fig. 6.

5 Test & Results

5.1 Validation of the Model

5.1.1 Consistency of Turbine Parameters

The first attempt in validating the model is to compare the turbine response with the performance test results which are included in measurements done back in 1993 [7]. This is done to ensure that the parameters entered into the turbine model (e.g., nominal power, flow rate, Mechanical efficiency) are consistent with the turbine data table values.

For validating the consistency of the turbine model a simple model as shown in Fig. 7 was created.

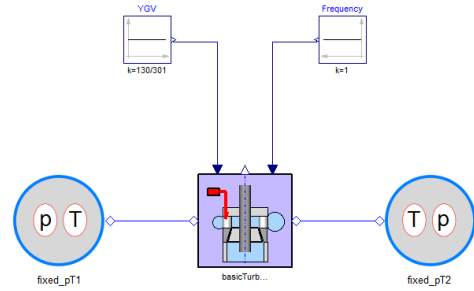


Figure 7: Model for simulation of the turbine consistency

This model is simulated for different guide vane openings given by the measurements from 1993 [7]. The frequency is kept equal to the nominal value in all of the simulations. In each simulation the constant pressure drop across the turbine is set to be equal to the value corresponding to the relevant guide vane opening.

5.1.2 Step Response Simulations

For this simulation the model shown in the Fig. 8 is used in which the input command is directly applied to the guide vane.

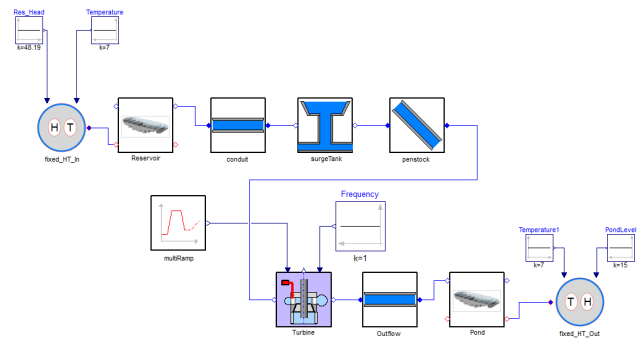


Figure 8: Test model for applying step changes in the guide vane input

Fig. 9 shows the step response of the model for a positive step change at time $t = 500\text{ sec}$ and then a negative step change applied at $t = 1500\text{ sec}$. For this simulation the relative pipe roughness of the conduit and the outflow tunnel is set to 0.065 as found from calculations. The relative pipe roughness of the penstock is set to 0.003. The guide vane servomotor opening time is limited to 20 sec (full range opening) and its closing

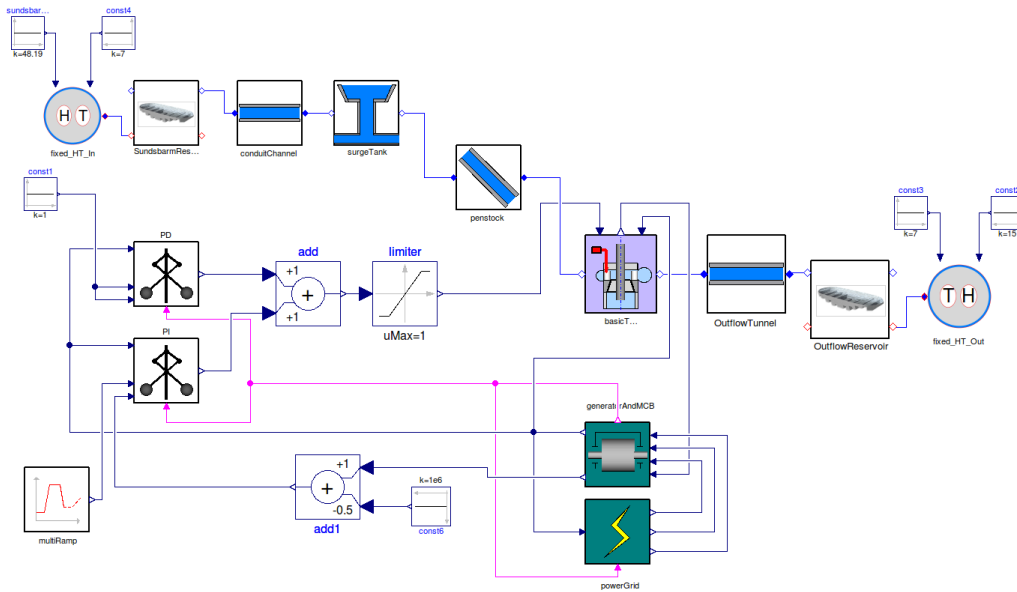


Figure 6: Complete model of Sundsbarm Power Plant modelled using the HydroPlant Library

time is limited to 2.5 sec. This effect can be seen in the guide vane opening in the Fig. 9. Because of this difference, the magnitude of the surge pressure at the turbine inlet is greater when the guide vane closes although the magnitude of negative change in guide vane opening is smaller.

5.2 Linearisation of the Model

Dymola can linearise nonlinear models around their steady state operating point. In this section a linear model will be obtained for the hydropower model that can be further analysed and used for design in the MATLAB environment. For this reason the model in Dymola needs some adjustments. Fig. 10 shows the resulting model.

After loading the steady state condition of the model in Dymola, a random noise input is used for linearising the model. The reduced order of the linearised state space model is 58.

5.2.1 Applications of the Linearised Model

The linearised model can be used for implementing more advanced control methods like the ones described in [8]. The model shown in Fig. 10 is specially tailored to be used for such control schemes:

- For saving number of state variables just fixed sources are used to model reservoirs.
- For having a time invariant dynamics, the load in the grid block is set to be constant and then the changes in the grid is modeled merely by adding a random disturbance signal to the grid balance.

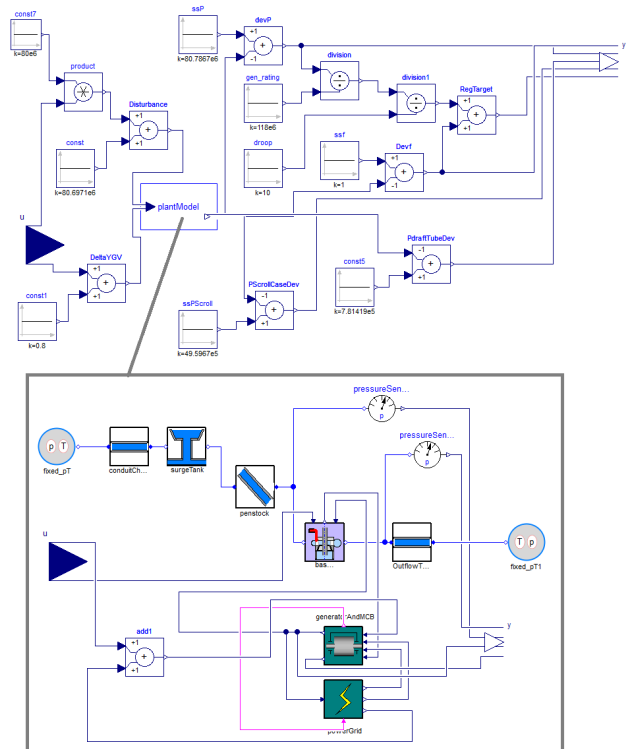


Figure 10: Adjustments done to the model before linearisation

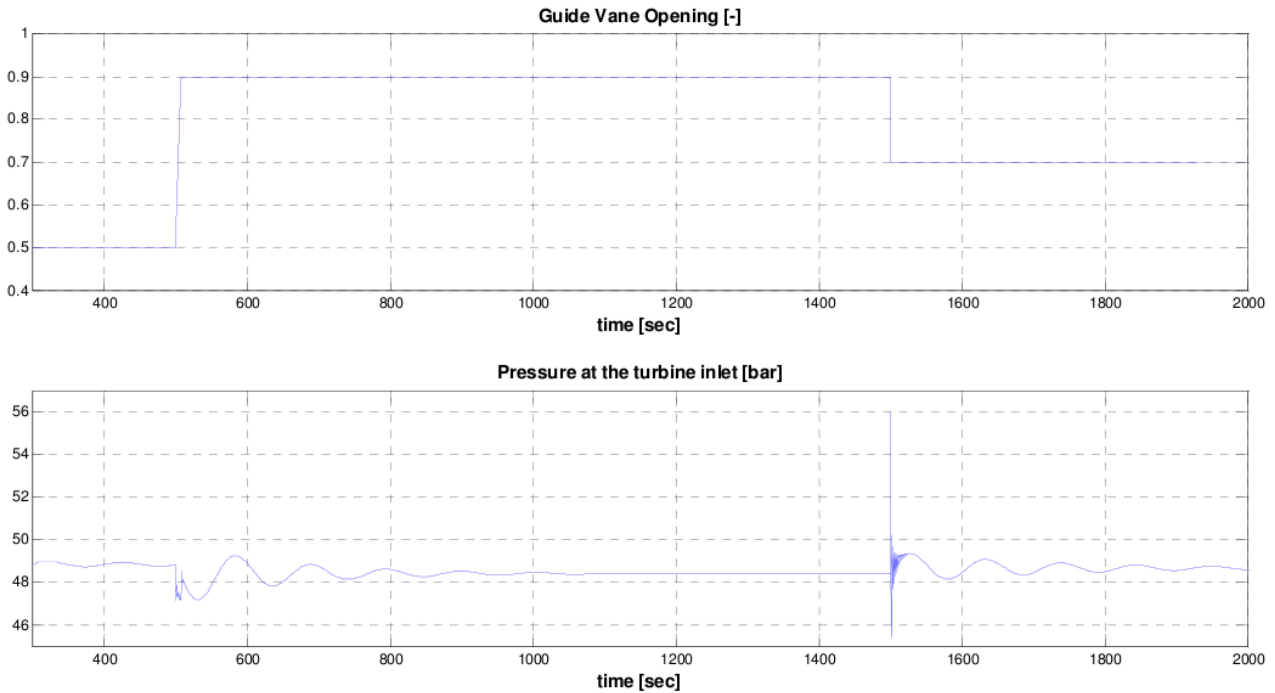


Figure 9: Step response (Relative pipe roughness of conduit and outflow tunnel=0.065 and Relative pipe roughness of penstock=0.003)

- The same opening and closing rate limits are considered for guide vane operation for better linearity. Additional rate for guide vane opening may be implemented in the controller.
- For having a strict constraint on the power generation/grid frequency values (to satisfy droop requirements) a combination of these variables are selected as output of the model and this combination can be used as a feedback for the controller.
- Other quantities (like pressure in different locations) can be selected as additional outputs to enforce constraints on the system state variables by controller. These constraints can be applied by advanced methods like Model Predictive Control. This can be considered as a future work.

6 Conclusions

In this paper the theory of hydro power systems was briefly presented. Based on dimensions and specifications, as well as structure drawings, (from Skagerak Energy's archive) a dynamic model of the Sundsbarm power station was created using the HydroPlant Library of Modelon AB.

At the time of writing only operational data of the

power station were available. Therefore the model was tested against friction values from pressure measurements at the conduit channel. Obviously with the more data points, a better model can be obtained. The data available do not cover the complete operational range of the guide van opening and the flow rate of the turbine. The relative wall roughness for the conduit was calculated from the measurement data. The model was then tested with the calculated roughness to verify that the same friction loss can be reproduced and the result was comparable. This means that steady state calculations in the HydroPlant Library are reliable.

Finally suggestion was made for implementation of a model predictive controller as part of a future work. A linear model had to be made in order to develop a MPC. The linear model had additional outputs for pressure in penstock and draft tube. Therefore a MPC controller could have these output constraints for pressure in penstock and the draft tube, which is believed that gives safer and more optimised control of the Sundsbarm power plant.

Other improvements that could have been implemented for the plant is to have power feedback for the turbine governor at the grid connection, instead at the generator terminals. This means correcting the power set point with measurements at the delivery point of the power to the grid. This way the power produc-

tion will have less deviation since the power readings for the grid operator are at the grid. This will allow the power plant to consider the loss of approx. 0.5 MW from the transformer, cables, and own power consumption, when adding a set-point for the plant.

7 Acknowledgements

This paper is based on the project report “Modeling and Optimization of Deviation in Hydro Power Production” within the course “SCE4006” at Telemark University College created by the students Hege Marie Thoresen, Ingvar Andreassen, Magamage Anushka Sampath Perera, and Behzad Rahimi Sharefi (also named as co-authors of this paper).

The project was done in cooperation with *Skagerak Energi* Porsgrunn, Norway. Many thanks to Jane Berit Solvi as contact person at *Skagerak Energi*.

References

- [1] C. Warnick, *Hydropower Engineering*. Prentice-Hall, Inc, 1984.
- [2] L. Arthur, “Lesotho highlands water project.,” in *ICE Proceedings, Civil Engineering .*, 1997.
- [3] Encyclopædia Britannica Online, “Synchronous Generator.” <http://www.britannica.com/EBchecked/media/1393/Elementary-synchronous-generator>, jan 2011.
- [4] A. Kjølle, “Hydropower in norway – mechanical equipment,” tech. rep., NTNU, Trondheim, Norway, 2001.
- [5] P. Kundur, *Power System Stability and Control*, vol. 1, ch. 9, p. 377ff. McGraw-Hill Inc., 1994.
- [6] D. Stuksrud, “System dynamics in Hydropower Plants,” tech. rep., NTNU, Trondheim, 1998.
- [7] Kværner-Energy, “Sundsborn Kraftverk Termodynamisk Virkningsgradsmåling,” tech. rep., Kværner-Energy, 1993.
- [8] N. Kishor, R. Saini, and S. Singh, “A review on hydropower plant models and control,” *Renewable and Sustainable Energy Reviews*, vol. 11, no. 5, pp. 776 – 796, 2007.

Thermal Separation Library: Examples of Use

Karin Dietl* Kilian Link† Gerhard Schmitz‡

Abstract

This paper deals with the Thermal Separation Library, which is intended to be used for absorption and rectification processes. Two example calculations show how the simulation speed can be increased by choosing the right way to set up the equations. One example refers to the ordering of the substances in the substance vector and one refers to the modelling of equilibrium processes. An example of use presented is the CO₂ absorption in a post-combustion carbon capture plant. The transient simulation results are compared to measurement data obtained in a Siemens pilot plant.

Keywords: thermal separation, carbon capture, absorption / desorption

1 Introduction

Dynamic analysis of thermal separation processes gains in importance, be it in batch processing, system control, start-up strategies or shut down behaviour in continuous processing ([9]).

General modelling approaches for these problems (or a part of these problems) have been reported in literature. READYS ([16]) as well as a model developed in [12] can be used for dynamic simulation of equilibrium columns. A model which considers dynamic simulation of multiphase systems is presented in [2]. [11] describes steady-state and dynamic non-equilibrium models. A tool for dynamic process simulation - DIVA - is proposed by [5], which has a sequential simulation approach. DYNAMSIM was developed by [3] and is a tool for design and analysis of chemical processes. In [13] a modelling language gPROMS is proposed to model combined lumped and distributed systems which was then successfully used in several publications in order to model separation processes (e.g. [23]).

The developed models have been tested on several different processes, mostly on a single column. There are also examples where more complex system designs are presented ([6]).

The aim of this paper is to describe a modelling library which can be used for flexible modelling and simulation of complex separation systems. As an example a fully integrated absorption/desorption loop for carbon capture is simulated. Simulation results will be compared to measurement data obtained in a Siemens pilot plant. The process is shown in figure 5 and will be discussed in more detail in section 4.1.

2 Modelling Approach

It is commonly agreed on that when developing a model of multicomponent system including chemical reactions, a simple equilibrium-based model which neglects mass transfer, is often not sufficient and a more physical approach - the rate-based approach ([18]) - is needed (e.g. [19], [17], [10]). The simulation models are built using the object-oriented Thermal Separation Library ([8]) which was developed in order to model dynamic absorption and rectification processes.

2.1 Model equations

2.1.1 Balance equations

The balance equations are established separately for the vapour and liquid bulk phase as described in [8]. There is therefore a mole balance for each component i of vapour and liquid phase. Additionally there should also be summation equations in the bulk phases ($\sum_i y_i = 1$ and $\sum_i x_i = 1$). However, in general systems of ordinary differential equations can be more easily solved than differential algebraic equations and introducing a differential equation instead of an algebraic equation is rewarded with faster computation times. Therefore these two algebraic equations are replaced by the total amount of substance balance for vapour and liquid phase.

For liquid and vapour phase there is one energy balance each (see [8]).

*Hamburg University of Technology, karin.dietl@tuhh.de

†Siemens, kilian.link@siemens.com

‡Hamburg University of Technology, schmitz@tuhh.de

Nomenclature		Greek symbols	
A	heat transfer area	$\varepsilon^{\text{vap}}, \varepsilon^{\text{liq}}$	vapour or liquid hold up
a	specific area	$[\Gamma]$	thermodynamic correction matrix
c	concentration	γ	activity coefficient
H	Henry coefficient	ϕ	fugacity coefficient
k	mass transfer coefficient	Subscripts	
M	molar mass	i	component i
\dot{N}	molar flow rate	j	stage j
n	number of discrete elements	r	reaction
n_S	no. of substances which are in both phases	Superscripts	
$n_S^{\text{liq}}, n_S^{\text{vap}}$	number of liquid / vapour substances	liq	liquid
$[R]$	rate matrix for mass transfer coefficients	sat	saturation
T	temperature	vap	vapour
u	specific inner energy	*	thermodynamic equilibrium
x	liquid composition	Abbreviations	
y	vapour composition	nLF	number of liquid feeds
Z	factor for equilibrium stage	nVF	number of vapour feeds

2.1.2 Rate Equations

In [8] the molar flow rates at stage j over the phase boundary are calculated as

$$\vec{N}_{i,j} = k_{i,j} \cdot a_j \cdot (c_{i,j}^{\vec{}} - c_{i,j}^*) \quad (1)$$

This approach however is only valid for binary systems, and using the concentration difference as driving force is only applicable for isothermic conditions. Therefore this was replaced by the more general Maxwell-Stefan equations as described by [22]:

$$\vec{N}_j^{\text{vap}} = \dot{N}_{\text{total},j}^{\text{vap}} \cdot \vec{y}_j + c_j^{\text{vap}} \cdot a_j \cdot [R_j^{\text{vap}}]^{-1} \cdot [\Gamma_j^{\text{vap}}] \cdot (\vec{y}_j - \vec{y}_j^*) \quad (2)$$

$$\vec{N}_j^{\text{liq}} = \dot{N}_{\text{total},j}^{\text{liq}} \cdot \vec{x}_j + c_j^{\text{liq}} \cdot a_j \cdot [R_j^{\text{liq}}]^{-1} \cdot [\Gamma_j^{\text{liq}}] \cdot (\vec{x}_j - \vec{x}_j^*) \quad (3)$$

The thermodynamic correction matrices $[\Gamma]$ are necessary, since here the difference of the mole fraction as driving force is used and not the difference in the chemical potential. The $[\Gamma]$ -matrices contain the composition derivatives of the activity coefficient (liquid) or fugacity coefficient (vapour), which can be found for several activity coefficient models in [21]. The $[R]$ -matrices are calculated using the binary mass transfer coefficients. How to obtain the $[R]$ -matrices via the binary mass transfer coefficients is described in detail in [22].

The total molar flow rates in (2) and (3) are obtained by summing the molar flow rates for each component.

It is also important to note that the vectors \vec{N}_j^{vap} and \vec{N}_j^{liq} in eq. (2) and (3) contain only the molar flow rate $\dot{N}_{j,i}$ of $n_S^{\text{vap}} - 1$ and $n_S^{\text{liq}} - 1$ substances respectively, that is this equation is not established for every substance. The missing equations are the summation equations at the phase boundary:

$$\sum_i y_i^* = 1, \quad \sum_i x_i^* = 1 \quad (4)$$

2.1.3 Inert Substances

An equation describing the thermodynamic equilibrium does only exist for the n_S components, which exist in both phases, but not for inert substances. The missing equations for the inert substances are obtained by setting the molar flow rate over the phase boundary of the inert substances to zero:

$$\left. \begin{array}{l} \dot{N}_{i,j}^{\text{vap}} = 0 \\ \dot{N}_{i,j}^{\text{liq}} + \dot{N}_{r,i,j}^{\text{film}} = 0 \end{array} \right\} \text{if substance } i \text{ is inert} \quad (5)$$

2.2 Numerical Solutions

In order to solve the resulting system of differential-algebraic equations using numerical integration, as many state variables as differential equations are needed. For the columns, the following variables are chosen as state variables: $c_{j,i}^{\text{vap}}, c_{j,i}^{\text{liq}}, u_j^{\text{vap}}, u_j^{\text{liq}}, \varepsilon_j^{\text{vap}}$ and T_j^{vap} (or M_j^{vap} which is as suitable), with $j = 1 \dots n$ and $i = 1 \dots n_S^{\text{vap}}$ or $i = 1 \dots n_S^{\text{liq}}$ respectively. Using this set

of state variables, all other variables can be calculated via algebraic relations. Intensive rather than extensive quantities were chosen as state variables, since the differential equations were set up using these intensive variables.

2.3 Library Structure

The library structure of the column models is represented in figure 1 which shows a class diagram of the library. The three different column types (packed column, tray column, spray column) all inherit from `FeedStage` (which is denoted with an arrow pointing at the parent class). The `FeedStage`-model inherits from the `BaseStage`-model which contains the balance equations and some of the constitutive equations for n discrete elements. It also contains instances of the medium models, the reaction models etc. These models are declared as replaceable, where replaceability is denoted using a dotted line with a diamond.

The extending column classes supply the geometry, the instances of the pressure loss and liquid holdup model, the heat transfer model between the two phases and the mass transfer models. Each column type is structured the same way; but only the structure of the packed column is shown in the diagram due to readability.

3 Examples

As stated in the introduction it should be possible to model and simulate a complex separation process using the Thermal Separation Library. In the following a complete absorption/desorption loop is presented. Before doing so some small examples show how a different writing of the modelling equations can increase simulation speed.

3.1 Ordering of Substances in Medium Model

In general any order of the substances in the medium model can be chosen. However since for a non-equilibrium model the equations for the molar flow rates eq. (2) and (3) exist only for $n_s^{\text{vap}} - 1$ and $n_s^{\text{liq}} - 1$ substances respectively, and the molar flow rate for the last substance in the medium model is defined via the summation equations (4), the ordering becomes important in case the fraction of the last substance becomes very small (in the order of the tolerance of the numerical solver). In this case the calculation can become very slow and the composition of the last sub-

stance at phase boundary may even be calculated to be negative. This is known as cancellation problem (see e.g. [4]). Figure 2 shows an example for the absorption of N_2 and O_2 in H_2O . The simulation time decreases by the factor 5 if not O_2 is the last substance in the medium model.

So it can be stated having substances with very low concentrations at the last place in the substance vector decreases simulation speed. If this is not possible (i.e. no substance is always large enough to be suitable as last component) it is also possible to apply the Maxwell-Stefan equations to all components and omit the summation equations at the phase boundary. This also reduces the CPU-time, but in some test cases the sums of the mole fractions at the phase boundary were observed to deviate about 10% from 1.

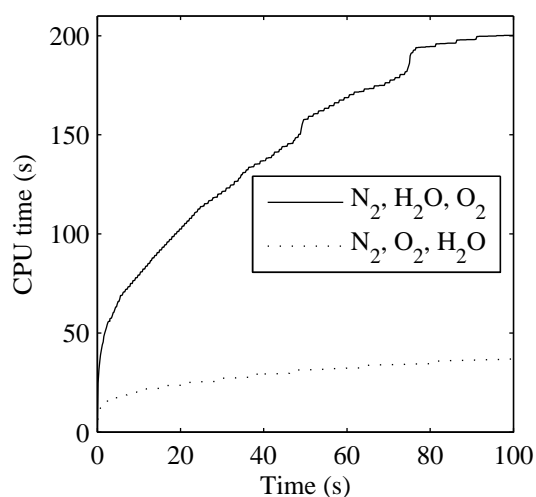


Figure 2: Influence of the ordering of the substances in the medium model. Simulation speed is increased, if in the liquid phase a substance with a high mole fraction (H_2O) is the last substance in the medium model (i.e. the CPU time for simulation is smaller).

3.2 Equilibrium Model

As stated in chapter 2 the stages were modelled as non-equilibrium stages, i.e. mass transfer is taken into account. However sometimes it is advantageous to describe the separation column using equilibrium stages, e.g. if no suitable correlations for the mass transfer are available.

There are basically two different approaches to model such an equilibrium stage: The first possibility is to implement a new set of equations which describe the equilibrium stage. However the additional algebraic constraint of the compositions at phase boundary and

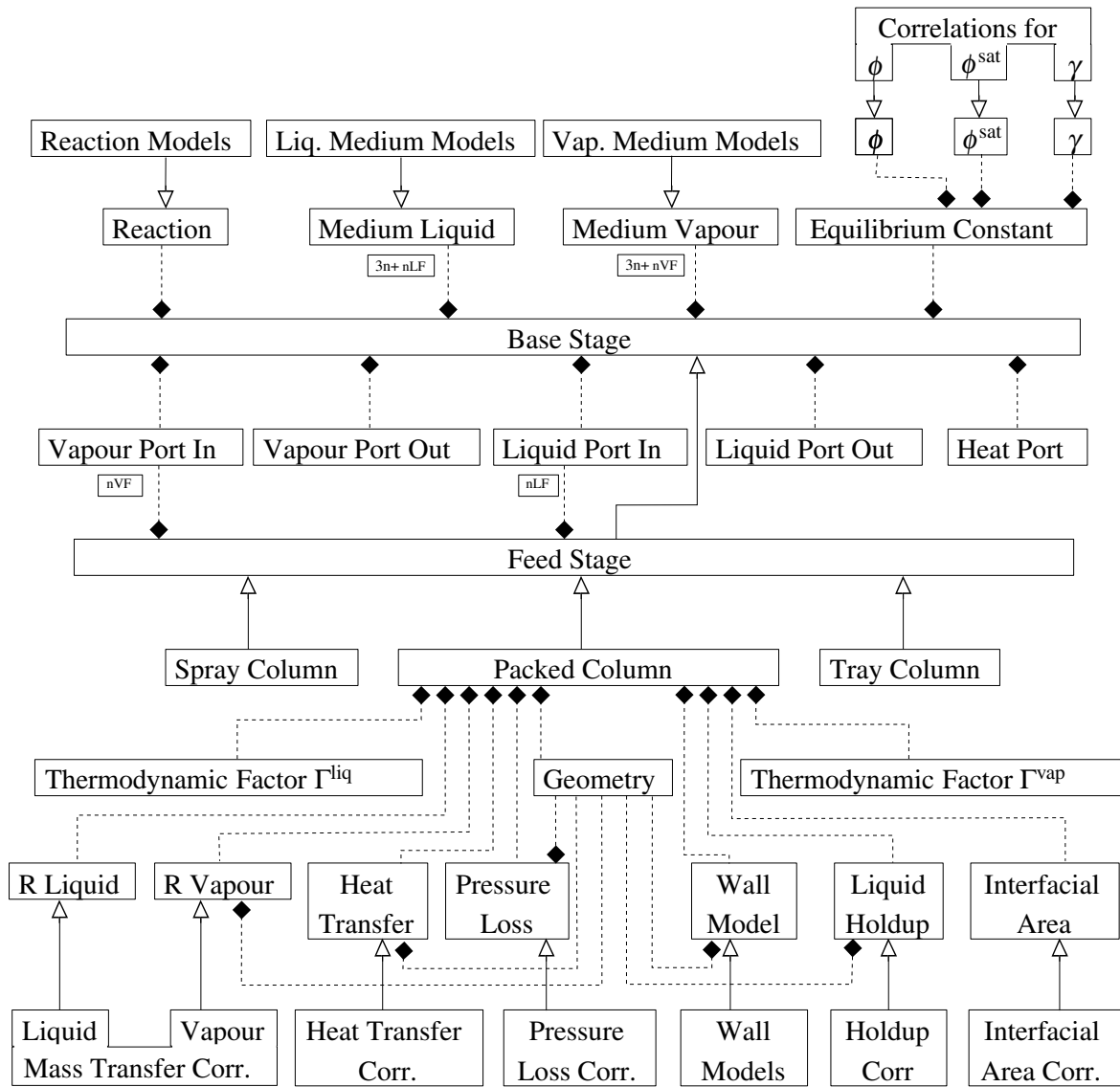


Figure 1: UML class diagram of an absorption or rectification process. The arrow denotes inheritance, i.e. FeedStage inherits from BaseStage only. The line with the diamond denotes composition. Dotted lines mean that the object is replaceable. The composition of spray column and tray column are analogue to the composition of the packed column.

in the bulk phase being equal result in a large nonlinear system of equations.

The second possibility is to approach the thermodynamic equilibrium ($x_i \rightarrow x_i^*$) using the non-equilibrium equations and increasing the binary mass transfer coefficients to infinity. Using this approach the system is less strongly coupled, but still it has to be noted that the equations (2) to (4) introduce a high non-linearity to the system. Together with a very high value for the binary mass transfer coefficients, necessary in order to approach equilibrium, the computation time can become very large (see figure 3). Since the equation (2) - (4) do not give any information for the equilibrium model they are replaced by the much more simpler eq. (6)-(7):

$$\dot{N}_{j,i}^{\text{vap}} = Z_{j,i}^{\text{vap}} \cdot (y_{j,i} - y_{j,i}^*) \quad (6)$$

$$\dot{N}_{j,i}^{\text{liq}} = Z_{j,i}^{\text{liq}} \cdot (x_{j,i} - x_{j,i}^*) \quad (7)$$

were Z is an adjustable factor. If Z approaches infinity, the difference between the composition at the phase boundary and in the bulk phase vanishes and equilibrium is attained. A very high value for Z lead to a very accurate result. An indicator for the accuracy is the difference of the composition at the phase boundary to the composition in the bulk phase. This difference should become zero for an equilibrium model. However for a very high value for Z , the computation time may become very large or - even worse - there are convergence problems with the nonlinear solver. Since a optimal value for Z , which leads to an acceptable compromise between computing time and accuracy is not constant during simulation, Z is continuously adapted to minimize the difference between bulk and phase boundary composition using the equations of a simple PI controller.

Equations (6) and (7) are then replaced by equations (8) and (9), where Z are functions of the error ($y_{j,i} - y_{j,i}^*$) and ($x_{j,i} - x_{j,i}^*$) respectively and some controller parameters which are constant during simulation.

$$\dot{N}_{j,i}^{\text{vap}} = Z_{j,i}^{\text{vap}}(y_{j,i} - y_{j,i}^*, \text{contr. parameter}) \cdot (y_{j,i} - y_{j,i}^*) \quad (8)$$

$$\dot{N}_{j,i}^{\text{liq}} = Z_{j,i}^{\text{liq}}(x_{j,i} - x_{j,i}^*, \text{contr. parameter}) \cdot (x_{j,i} - x_{j,i}^*) \quad (9)$$

Figures 3 and 4 show that using eq. (6) and (7) (solid line) or eq. (8) and (9) (dotted line) instead of eq. (2), (3) and (4) (dashed line) lead to a lower computation time and a higher accuracy of the result. Adapting the variable Z during the simulation using a PI controller (dotted line) further increases accuracy

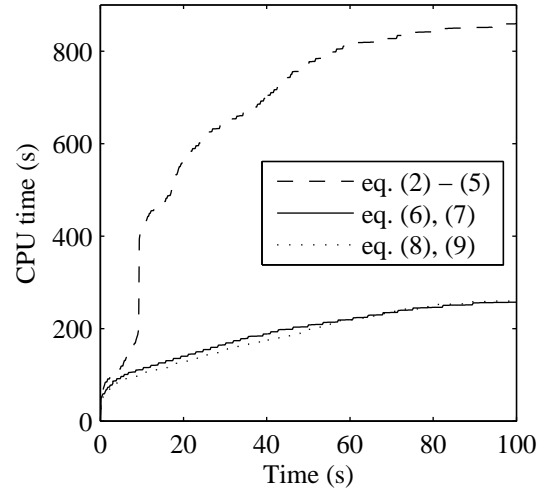


Figure 3: Computation time using three different sets of equations to model an equilibrium stage. The first set uses exactly the same equations as for a non-equilibrium model (eq. (2) to (4)) with a constant, but high mass transfer coefficient. The second and third set use simplified yet sufficient detailed equations which increases simulation speed.

without increasing the computation time, compared to eq. (6) and (7) where Z is constant.

In this example, the first set of equations has more time states than the last two (178 scalars instead of 162 scalars; for eight stages) but about the same amount of time varying variables (around 12000). The size of the nonlinear system of equations obtained by Dymola is higher, for the first set of equations: eight blocks of 20 iteration variables are necessary instead of eight blocks of 15 iteration variables (the other blocks are identical with lesser iteration variables).

The same approach is also chosen when reaction equilibrium should be assumed: in this case the controller minimizes the difference between the reaction equilibrium constant and the product of the activities.

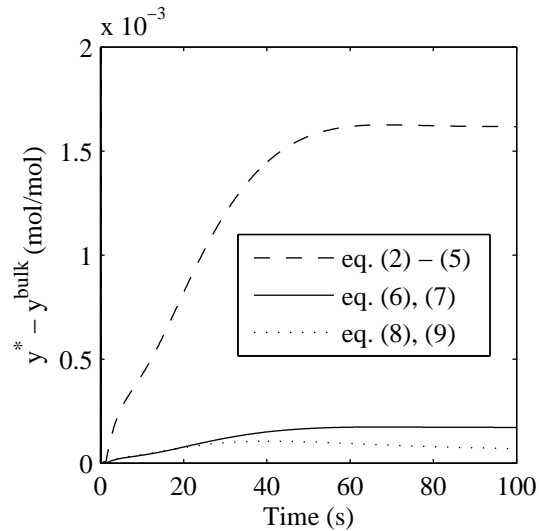


Figure 4: Largest difference in $y^* - y$ using three different sets of equations to model an equilibrium stage.

4 Dynamic Analysis of CO₂ Capture Plant

4.1 Plant Layout and Data

The simulation of the carbon capture plant refers to the same pilot plant as presented in [7]. It is a slip-stream pilot plant operating under real conditions. The absorber has a diameter of approx. DN200 and the absorber height is approx. 35 m. The plant layout is shown in figure 5.

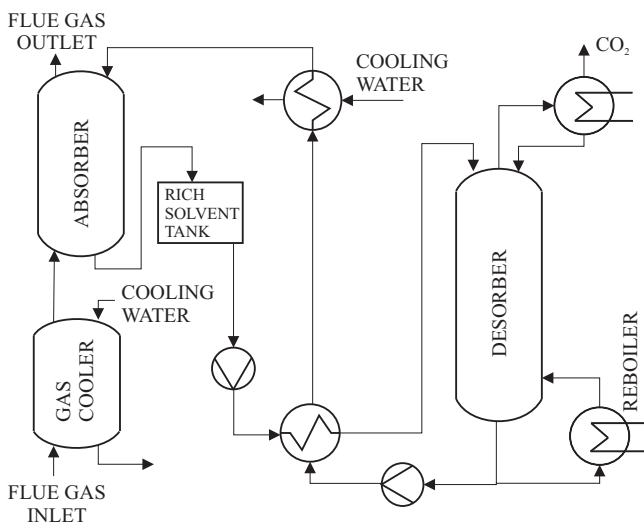


Figure 5: Carbon capture pilot plant as it can be found in [7].

The plant consists basically of a flue gas cooler, an absorber and a desorber (see figure 5). The flue gas first enters the flue gas cooler, where it is cooled down

Table 1: Physical properties and other parameters

Property	Reference
Pressure loss vapour	[20]
Interfacial heat transfer coeff.	Chilton-Colburn
Mass transfer coefficient	[15]
Liquid holdup	[14]
Diffusion coeff. vapour	[1]
Interfacial area	[15]

using pure water. In the absorber, the saturated flue gas is then brought in contact with a liquid containing a high amount of dissolved amino acid salt. Here, the CO₂ is absorbed by the liquid and nearly CO₂-free gas leaves the absorber. The loaded liquid is now heated up in a heat exchanger before it enters the desorber, where the CO₂ is stripped from the liquid using water vapour, which is obtained in a thermosyphon. The gas leaving the desorber (containing water and CO₂) is cooled down and such water is removed via condensation. The pure CO₂ is then liquified and stored; the pure water is fed back to the desorber. The now unloaded liquid from the desorber bottom is partly evaporated in a thermosyphon and partly led back to the absorber via two heat exchangers in order to cool down to the absorber temperature.

The solvent used is an amino-acid salt solution, as described in [7]. The underlying chemical reaction is shown in figure 6. In the model, it is assumed that reaction takes place in the film only and reaction kinetics are not considered.

Table 1 provides an overview of the most important physical properties.

4.2 Thermodynamic equilibrium

4.2.1 Phase equilibrium CO₂

A very simple approach proposed by Siemens was chosen to model the phase equilibrium of CO₂ between liquid and gas phase. This approach combines the dissolution of CO₂ in the liquid and the stepwise reaction with the dissolved salt to the final product in a single phase equilibrium equation. The main advantage is that only very few components in the liquid phase have to be modelled, namely H₂O, the dissolved salt called AAS (amino acid salt) and the final reaction product. All reaction intermediate products as well as dissolved molecular CO₂ in the liquid phase have not to be considered, the latter due to the

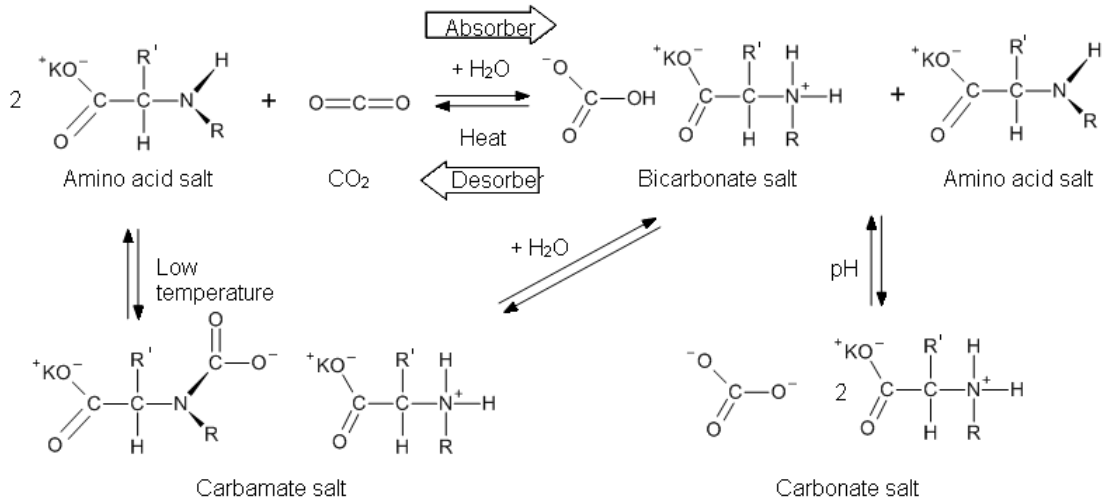


Figure 6: Reaction scheme in the amino acid salt solution, as presented in [7]

assumption that the reaction is nearly instantaneous and the CO_2 concentration in the liquid is negligible. This means that if a molar flow rate \dot{N}_{CO_2} of gaseous CO_2 crosses the phase boundary, a source term increases the molar quantity of the reaction product and a sink term decreases the molar quantity of the educts (H_2O , AAS) such accounting for the stoichiometrics.

Since CO_2 is nonexistent in the model of the liquid phase the phase equilibrium is described using the CO_2 vapour mole fraction on the one side and the mole fraction of the reaction product on the other side:

$$p \cdot y_{\text{CO}_2} = \frac{x_{\text{Product}}}{1 + x_{\text{Product}}} \cdot H(T) \cdot \gamma(T, x_{\text{Product}}) \quad (10)$$

The Henry coefficient H is temperature dependent and the activity coefficient γ is additionally dependent on the concentration of the product in solvent. For both coefficients Siemens provides a correlation, which are as follows:

$$\ln H = a + \frac{b}{T} \quad (11)$$

$$\ln \gamma = e^{(c+d \cdot T + e \cdot T^2)} \cdot \frac{x_{\text{Product}}}{1 + x_{\text{Product}}} \quad (12)$$

The parameters a , b , c , d , e have been adjusted using measurement data.

It has to be stated that this approach has drawbacks concerning the accurateness of the results, nevertheless they are all in the right order of magnitude and the main factors influencing the result are considered. This approach is therefore suitable for dynamic simulation, where the dynamics are in the focus of the study and a high simulation speed is needed but very correct steady-state results are not of such importance.

4.2.2 Phase equilibrium water

The water equilibrium is calculated using

$$p \cdot y_{\text{H}_2\text{O}} = x_{\text{H}_2\text{O}} \cdot p^{\text{sat}} \cdot \gamma \cdot \phi^{\text{sat}} \quad (13)$$

For the calculation of the water saturation pressure, the saturation pressure of pure water is multiplied by a factor smaller 1 to match the higher saturation temperature. The activity coefficient γ is set equal to one, even though it should be smaller than one in reality. At Siemens steady-state calculations were performed using AspenPlus where much more detailed medium models than the ones here were used. These AspenPlus calculations revealed that the activity coefficient did not differ significantly from one in the considered temperature and concentration range. Therefore, since the proposed modelling of the thermodynamic equilibrium is anyway not suitable in order to obtain very accurate steady-state results, setting the water activity coefficient equal to one does not increase the overall error very much.

4.3 Medium models

4.3.1 Gas mediums

Both gases, the flue gas, consisting of N_2 , CO_2 , O_2 and H_2O and the CO_2 -water vapour mixture in the desorber, are modelled as ideal gas, even though this assumption is more questionable for the CO_2 -water vapour mixture, due to the high water vapour content (about 70%-90%) at the saturation temperature of water (around 100°C). However the error due to differences in density and enthalpy are not important.

4.3.2 Liquid mediums

The pure water in the flue gas cooler is modelled using the waterIF97 medium model from the Modelica Standard Library.

The solvent is also modelled based on the water IF97 standard, but density and heat capacity are adjusted to match the solvent values (for a solvent containing approx. 30 weight-% AAS). All solvent medium properties are temperature and in parts also pressure dependent, but independent of composition.

As stated in 4.2.2 the solvent model consists only of three components in order to increase simulation speed.

4.4 Simulation results

In this section the simulation results are presented and compared to measurement data obtained in the Siemens pilot plant.

Two different test cases are investigated:

- Test case 1: Increase of Flue gas flow rate from 75% to 100% at $t = 0$ min.
- Test case 2: Decrease of steam flow rate at re-boiler from 90% to 75% at $t = 0$ min.

Figure 7 - 9 show the comparison between simulation and measurements of test case 1. All temperatures are plotted referring to an arbitrary reference temperature. The CO_2 mass flow rate is described in percent, whereas 100% is arbitrarily set to the measurement value obtained at $t=0$ min. It can be seen that the steady-state results differ, which is due to the very simplified modelling of the thermodynamic equilibrium. The dynamic response between simulation and measurement are quite similar, even though in the simulation the new steady-state is obtained after a shorter time.

For test case 2 the comparison between simulation and measurements are shown in figure 10 - 14. Again, all temperatures are plotted referring to an arbitrary reference temperature. As for test case 1 the steady-state results differ, but the dynamic behaviour is similar. In figure 12 not only the measured CO_2 mass flow rate at the desorber outlet is plotted, but also the mass flow rate of the absorbed CO_2 in the absorber. The latter value was obtained from measurement data, using measured mass flow rates, temperatures and volume fractions at the absorber in- and outlet. These two mass flow rates should be equal in steady-state (which

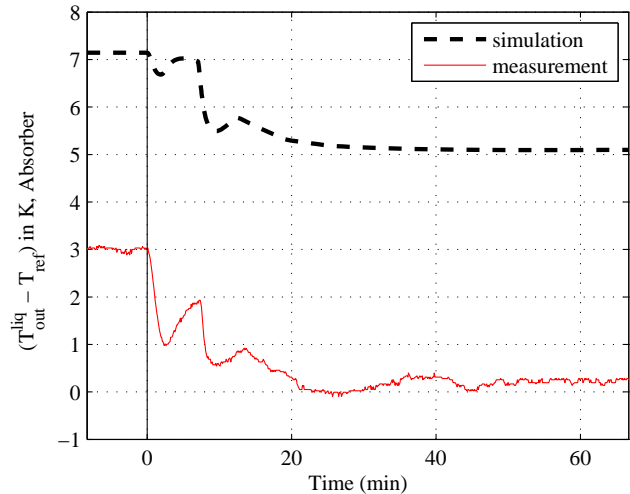


Figure 7: Liquid outlet temp., absorber (test case 1)

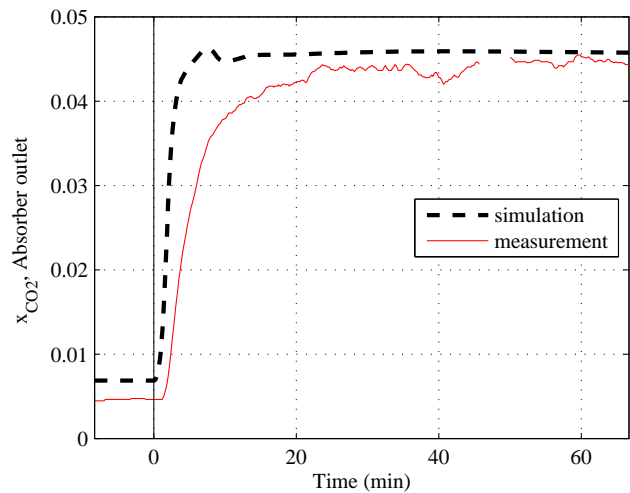


Figure 8: CO_2 outlet mole fraction (test case 1)

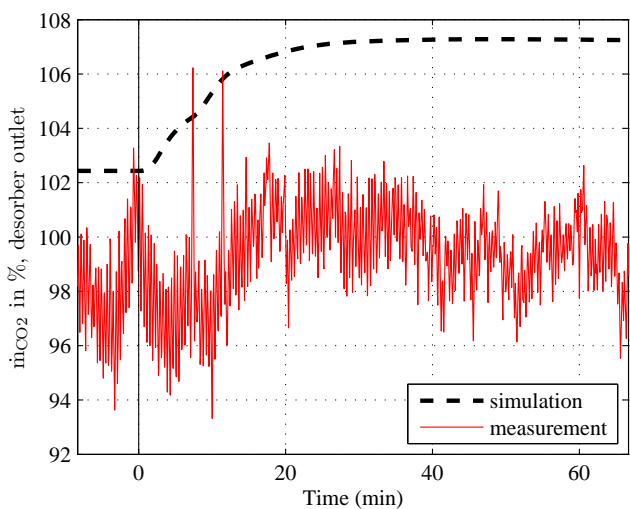


Figure 9: CO_2 mass flow rate, desorber (test case 1)

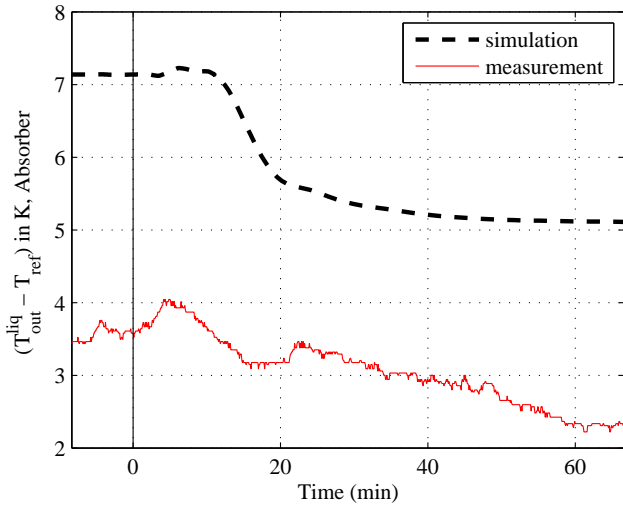


Figure 10: Liquid outlet temp., absorber (test case 2)

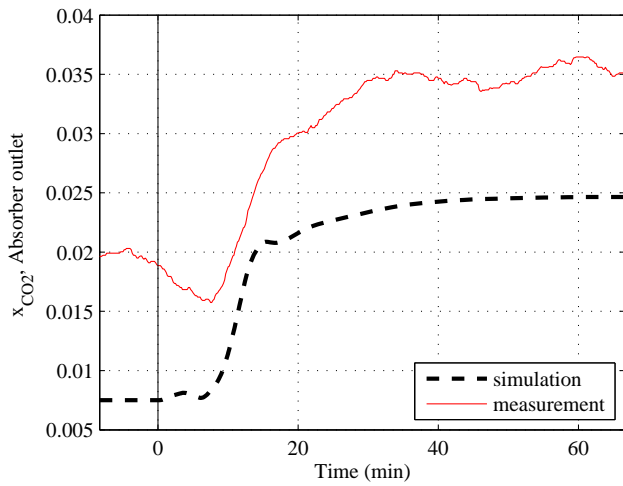


Figure 11: CO₂ outlet mole fraction (test case 2)

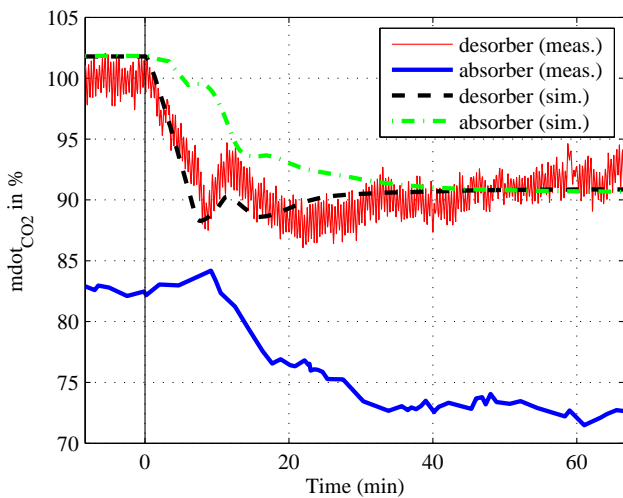


Figure 12: CO₂ mass flow rate (test case 2)

was the case in test case 1), however there is a difference of about 20%. Since this difference does not occur in the simulation, it explains why measurement and simulation fit well for the CO₂ mass flow rate at the desorber outlet, but not for the CO₂ mole fractions in the absorber and consequently not for the amount of CO₂ absorbed.

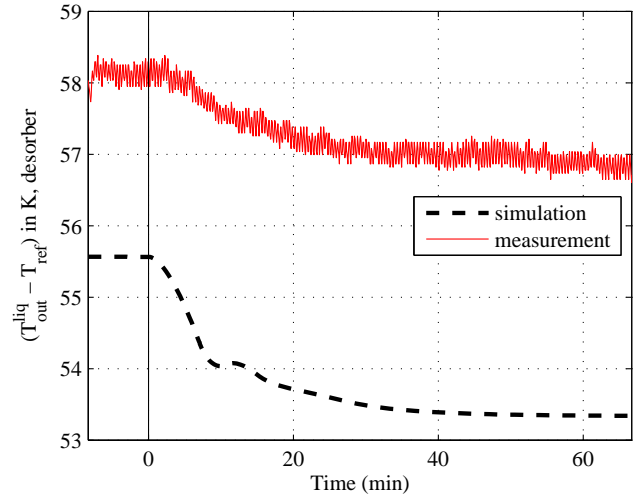


Figure 13: Liquid outlet temp., desorber (test case 2)

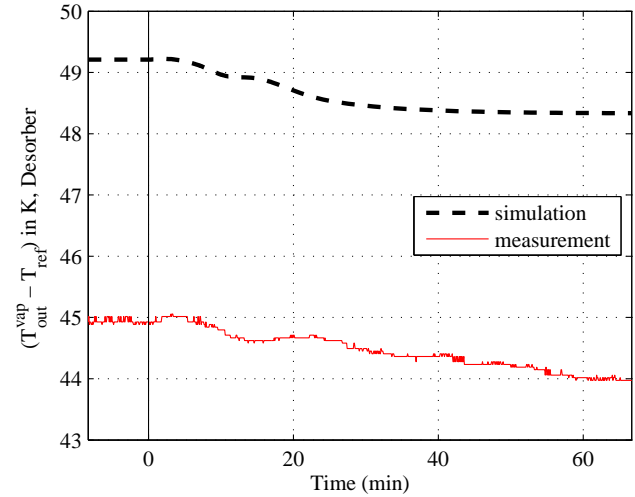


Figure 14: Vapour outlet temp., desorber (test case 2)

5 Summary

This work proposed a Modelica-library model for dynamic simulation of tray and packed columns for separations processes such as absorption and rectification. It is shown that it is advantageous to describe an equilibrium model using modified equations of the non-equilibrium model. Also the influence of the ordering

of the substances in the medium vector was shown. It was also shown that using the Thermal Separation Library a complex system, namely an absorption/desorption loop for carbon capture is modelled and simulated dynamically. The simulation results were compared to measurement data obtained in a Siemens pilot plant. It showed good agreement, even though there were differences in the stationary results which is due to the very simplified modelling of the thermodynamic equilibrium.

References

- [1] Edward N. Fuller, Paul D. Schettler, and J. Calvin Giddings. A new method for prediction of binary gas-phase diffusion coefficients. *Industrial and Engineering Chemistry*, 58(5):19–27, May 1966.
- [2] R. Gani, Thomas S. Jespen, and Eduardo S. Perez-Cisneros. A generalized reactive separation unit model. modelling and simulation aspects. *Computers Chemical Engineering*, 22(Supplement):363–370, 1998.
- [3] R. Gani, Esben L. Sorensen, and Jens Perregaard. Design and analysis of chemical processes through DYNsIM. *Ind. Eng. Chem. Res.*, 31:244–254, 1992.
- [4] David Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys*, 23(1):5–48, March 1991.
- [5] P. Holl, W. Marquardt, and E. D. Gilles. DIVA - a powerful tool for dynamic process simulation. *Computers chem. Engng*, 12(5):421–426, 1988.
- [6] Bernhard Hüpen and E. Kenig. Rigorose modellierung und simulation von chemisorptionsprozessen. *Chemie Ingenieur Technik*, 77(11):1792–1798, 2005.
- [7] Tobias Jockenhoevel, Ruediger Schneider, and Helmut Rode. Development of an economic post-combustion carbon capture process. *Energy Procedia*, 1:1043–1050, 2009.
- [8] Andreas Joos, Karin Dietl, and Gerhard Schmitz. Thermal separation: An approach for a modelica library for absorption, adsorption and rectification. In Francesco Casella, editor, *Proceedings of the 7th International Modelica Conference*, Linköping Electronic Conference Proceedings, pages 804–813. Linköping University Electronic Press, September 2009.
- [9] E. Kenig. Complementary modelling of fluid separation process. *Chemical Engineering Research and Design*, 86:1059–1072, 2008.
- [10] E. Kenig, Kaj Jakobsson, Peter Banik, Juhani Aittamaa, and Andrzej Gorak. An integrated tool for synthesis and design of reactive distillation. *Chemical Engineering Science*, 54:1347–1352, 1999.
- [11] Hendrik A. Kooijman. *Dynamic Nonequilibrium Column Simulation*. PhD thesis, Clarkson University, 1995.
- [12] J. M. Le Lann, J. Albet, X. Joulia, and B. Koehret. A multipurpose dynamic simulation system for multicomponent distillation columns. *Computer Applications in Chemical Engineering*, pages 355–359, 1990.
- [13] M. Oh and Constantinos C. Pantelides. A modelling and simulation language for combined lumped and distributed parameter systems. *Computers chem. Engng*, 20(6/7):611–633, 1996.
- [14] J. Antonio Rocha, J. L. Bravo, and J. R. Fair. Distillation columns containing structured packings: A comprehensive model for their performance 1. hydraulic models. *Ind. Eng. Chem. Res.*, 32:641–651, 1993.
- [15] J. Antonio Rocha, J. L. Bravo, and J. R. Fair. Distillation columns containing structured packings: A comprehensive model for their performance. 2. mass transfer model. *Ind. Eng. Chem. Res.*, 35:1660–1667, 1996.
- [16] C. A. Ruiz, M. S. Basualdo, and N. J. Scenna. Reactive distillation dynamic simulation. *Institution of Chemical Engineers*, pages 363–378, 1995.
- [17] M. Schenk, R. Gani, D. Bogle, and E. N. Pistikopoulos. A hybrid modelling approach for separation systems involving distillation. *Trans IChemE*, 77:519–534, 1999.
- [18] J. D. Seader. The rate-based approach for modeling staged separation. *Chemical Engineering Progress*, pages 41–49, 1989.

- [19] M. S. Sivasubramanian and Joseph F. Boston. The heat and mass transfer rate-based approach for modeling multicomponent separation processes. *Computer Applications in Chemical Engineering*, pages 331–336, 1990.
- [20] J. Stichlmair, J. L. Bravo, and J. R. Fair. General model for prediction of pressure drop and capacity of countercurrent gas/liquid packed columns. *Gas Separation & Purification*, 3:19–28, March 1989.
- [21] Ross Taylor and Hendrik A. Kooijman. Composition derivatives of activity coefficient models. *Chem. Eng. Comm.*, 102:87–106, 1991.
- [22] Ross Taylor and R. Krishna. *Multicomponent mass transfer*. John Wiley & Sons, Inc., 1993.
- [23] M. L. Winkel, L. C. Zullo, P. J. T. Verheijen, and Constantinos C. Pantelides. Modelling and simulation of the operation of an industrial batch plant using gPROMS. *Computers chem. Engng*, 19:571–576, 1995.

Scalable-detail modular models for simulation studies on energy efficiency

Marco Bonvini*, Alberto Leva

Dipartimento di Elettronica e Informazione, Politecnico di Milano
Via Ponzio 34/5, 20133 Milano, Italy
{bonvini,leva}@elet.polimi.it

*PhD student at the Dipartimento di Elettronica e Informazione

Abstract

Simulation is widely used to assess and/or improve the energy efficiency of both existing and new buildings. Such an analysis has to account for heterogeneous phenomena efficiently, to manage components in a modular manner, and (which is seldom addressed in a structured way) to scale the detail level in all or part of the model, based on the particular simulation goal. In this manuscript, a proposal is formulated on how to structure a Modelica library so as to satisfy such a need.

Keywords: Building simulation; energy optimisation; object-oriented modelling; modular modelling; scalable detail.

1 Introduction

At each step of the design or refurbishing of a building, decisions need taking, based on some goal and on the state of the project, i.e., the decisions taken in the past. Like any engineering process, building (re)design is in fact a cyclic activity, where any choice has to be reconsidered when its effects – no matter how later observed – are found to be unsatisfactory.

Most of the mentioned decisions are complex, however, and to gather the necessary information for them, simulation is often the only viable way to go. In an ideal world, a simulation model should thus be available on the engineer’s desk throughout the project, ready to help for complex decisions like a pocket calculator helps for simple computations. It should be possible to simulate the project at any time, irrespective of what was already fully designed, and what conversely was only specified in terms of the boundary conditions provided for the rest of the overall system. It should also be possible to move back and forth among the complexity levels implicitly defined above, in the case some past decision needs re-discussing.

Moreover, as the project moves toward its maturity – and the model becomes correspondingly complicated – one should still have the possibility of replacing parts of said model with simpler ones, so as to improve simulation speed when only some aspects of the building’s behaviour need investigating. And of course, the effort required to create the simulation model and keep it aligned to the project must be reasonable, i.e., adequately compensated by design quality improvements.

Such an approach to simulation is very different from those adopted by typical engineering tools. Most are domain-specific (e.g., electrical, Energy System or ES, Computational Fluid Dynamics or CFD, and so forth), or have limited flexibility (e.g., there is a library of pre-built “boiler” models and creating a new one is very far from trivial), or do not allow for a structured management of the models and simulations within a project, or any combination thereof. Needless to say, adopting the Object-Oriented Modelling and Simulation (OOMS) paradigm, and in particular the Modelica language, is a very promising idea.

In fact, several Modelica libraries for building simulation already exist [1, 2, 5]. However, the use of such libraries as a decision aid *along* the evolution of a project still experiences some difficulties. This work presents the authors’ opinion on the matter, and proposes a possible *modus operandi* to solve the encountered problems.

2 Problem statement

Traditionally, the (re)design of a building is treated as the partially disjoint (explanations follow) design of its “subsystems”. Although there is no standardised nomenclature, in fact, virtually the totality of engineering tools broadly distinguish (a) the “building” *stricto sensu* or “containment”, i.e., walls, doors, win-

dows and so on, (b) the contained air volumes, possibly divided in zones, (c) the Heating, Ventilation and Air Conditioning (HVAC) system, (d) automation and control systems, and (e) energy sources/sinks owing to the building utilisation, e.g., the heat released by occupants, industrial machines, or whatever is installed. The subsystems' interaction is accounted for by having some of them provide boundary conditions for the design of some other.

This is apparently very far from a really integrated approach, whence the term “partially disjoint” applied above to current design practices, but tools that address the simulation of all (or at least part) of the subsystems in a coordinated way are at present little more than research objects [3, 5, 4].

There is more than one reason for such a *scenario*. The most widely acknowledged one is given by the very different issues posed by the various subsystems. For example, control system models are made of oriented blocks and may need sometimes a continuous-time and sometimes a digital representation depending on the simulation purpose; models for HVAC, conversely, live invariantly in the continuous-time domain, but are typically zero- or one-dimensional, while models of phenomena that occur in *continua* such as a wall or an air volume often cannot avoid three-dimensional spatial distributions.

However, at least another reason needs mentioning. During its design, a building is looked at by various professionals, each one considering one or a few subsystems, and adopting a specific schematisation, ranging from 2D or 3D CAD drawings to piping diagrams, electrical schemes, and so forth. Apparently none of those schematisations is suitable for system-level modelling, which means that some new ones need introducing—whence a further difficulty.

Moreover, the designed diagrams tend to reach their final detail in a very few steps: for example a heating system may be specified as a P&ID, but then it is typically drawn in its complete layout, and more or less same is true for structures, walls, shadings, and so on.

As any expert knows, the development and maintenance of a simulation model follows a completely different path, especially if the model is conceived as a design decision aid. It *must* not be necessary to know much building details *before* being able to perform the *first* simulation, contrary to what one may be led to think, based on how most Modelica libraries on this matter (including those developed by the authors, of course) are structured.

In synthesis, our opinion is that structuring a Mod-

elica library for building simulation as a decision aid, is better done based on the *detail levels* one needs throughout a study. It should be stressed, for the sake of clarity, that we are dealing with the structuring of a *library*, not (necessarily) of models built from it. The aim is to facilitate the construction of said models in the most effective way to follow the project cycle. Of course, after such a structuring, most of the connector abstraction work will go on in the traditional way, but the aspect just mentioned remains the key one.

3 A library structuring proposal

As anticipated, simulation-based analysis needs conducting at different levels of detail. This remark can lead to a library structuring, which we propose to carry out in three steps.

3.1 Step 1

The first step is to define and qualify the mentioned detail levels. In this work we define four ones, corresponding to the basic questions encountered along a building project. Of course the matter is more articulated, and one could consider defining more levels, or further customising them based on the needs of some particular class of applications. For each defined level, we point out

- the purpose, i.e., what type of analysis it is conceived for;
- the hypotheses under which its models are valid;
- the analysis protocol, i.e., how the intended analysis is to be performed;
- the structural limitations, i.e., what facts the models are by construction unable to capture, and thus are implicitly considered neglectable in the intended analysis;
- the practice-based limitations, i.e., for example, what the models could in principle represent, but it is not convenient/cost-effective to have represented;
- and finally the (main) decision-making usefulness of the models.

Level 0

Purpose: determine/verify the overall first-cut energy needs on a static basis.

Hypotheses: the (single) internal air temperature follows the prescribed, constant set point; thermal capacities are disregarded; external ambient conditions are fixed; air renovation and exogenous energy sources are fixed based on the assumed utilisation.

Analysis protocol: a (static) simulation is done for each relevant *scenario* (e.g. a best and a worst case are defined for each climatic period in a year) and then results are combined in a straightforward manner.

Structural limitations: no dynamic phenomenon (due e.g. to heat storage) is accounted for, the source of the required energy is not discussed, no cost model is correspondingly introduced.

Practice-based limitations: it is generally inconvenient to introduce at this stage detailed models of the building containment (e.g., shading devices), whence a further source of approximation.

Decision-making usefulness: first overall assessment of the energy needs; possibility of evaluating high-level alternatives (e.g., it is already possible to roughly estimate the benefits of a certain type of insulation).

Note, incidentally, that level 0 is similar to that of (basic) energy certification analyses.

Level 1

Purpose: determine the overall energy needs accounting for internal thermal zones and heat storages in the containment.

Hypotheses: same as level 0 but with various internal air zones' temperatures, that follow the prescribed set points (here not constant) possibly filtered through some low-order dynamics to account for the control system's action, or at most with simplified descriptions of local controls; also, containment thermal capacities are considered.

Analysis protocol: same as level 0 except that here simulations are apparently dynamic.

Structural limitations: here too the source of the required energy is not considered (i.e., only the energy need is modelled, irrespective of the used mix of available sources), and no cost model is introduced.

Practice-based limitations: at this stage it can make sense to use detailed models of the building containment, while precise hypotheses on the control system's behaviour may be premature.

Decision-making usefulness: dynamic assessment of the energy needs, and possibility of evaluating high-level alternatives also regarding energy storages (e.g. the slower thermal behaviour typically induced by insulation is evidenced, and the temperature set point profiles can be discussed accordingly).

Level 2

Purpose: size/design/assess the energy system (ES) and discuss the energy mix.

Hypotheses: same as level 1 but air zones' thermal capacities are considered and the zone-level control system is introduced, including a reasonably detailed description of its physical realisation.

Analysis protocol: same as level 1.

Structural limitations: here the energy sources come into play but no detailed model of the generating devices (e.g. boilers) is used yet.

Practice-based limitations: at this stage reasonably detailed models of both the building containment and the zone-level control system are advised, while hypotheses on the energy sources are still coarse.

Decision-making usefulness: dynamic assessment of the ES and the zone-level controls capability of fulfilling the energy needs, including the discussion of possible alternatives (e.g. for the control system structuring and the energy mix) assuming an ideal behaviour of the energy sources.

Level 3

Purpose: size/design/assess the energy sources and the integrated control system, possibly including costs

Hypotheses: same as level 2 but more detailed models of the energy sources, and possibly the central controls, are introduced.

Analysis protocol: same as level 2.

Structural and practice-based limitations: conceptually this is the most detailed model possible with the available information, the only limitations come from errors in said information.

Decision-making usefulness: dynamic assessment of the integrated central and zone-level controls, possible optimisation of the set point curves based on cost considerations.

3.2 Step 2

The second step is to observe that the same detail levels above can be viewed from the model components' standpoint, resulting in the definition of which phenomena to represent, and how, in each of them. A synthetic list is given below.

Level 0

Containment elements: thermal conductances, possibly computed based on stratigraphies; correlations for solar radiation captation and exchanges with air/sky/terrain.

Internal air: a single prescribed temperature (*scenario*-based).

External ambient condition and solar radiation: prescribed (*scenario*-based).

Air renovation: prescribed flow rates (*scenario*-based).

ES: absent.

Exogenous energy sources (e.g. from machines, inhabitants, and so forth): fixed powers (*scenario*-based).

Control system: absent.

Level 1

Containment elements: same as level 0 but thermal capacities are introduced.

Internal air: a prescribed temperature per zone, possibly dynamically filtered (*scenario*-based), or some very simple description of local controls (but not of their physical realisation).

External ambient condition and solar radiation: same as level 0.

Air renovation: same as level 0.

ES: absent.

Exogenous energy sources: prescribed powers variable in time (*scenario*-based).

Control system: *de facto* absent if its action is summarised in the set point filters' time constants, or extremely simplified, see above.

Level 2

Containment elements: same as level 1.

Internal air: thermal capacities (possibly Mollier-based descriptions if humidity needs considering).

External ambient condition and solar radiation: same as level 1.

Air renovation: governed by the control system.

ES: piping and HVAC elements present, energy sources assumed to behave ideally (e.g. a boiler delivers the required flow rate at the required temperature).

Exogenous energy sources: same as level 1.

Control system: zonal controls represented, central ones idealised (in accordance with the partial ES representation).

Level 3

Containment elements: same as level 2.

Internal air: same as level 2.

External ambient condition and solar radiation: same as level 2.

Air renovation: same as level 2.

ES: same as level 2 but models for the energy sources are introduced.

Exogenous energy sources: same as level 2.

Control system: both central and zonal controls represented.

3.3 Step 3

The final step is to structure the library so that each component, preserving the physical interfaces, be described by different models depending on the required detail level. For example, in the following, wall or air models have the same connectors, but their equations change with the detail level, while the energy system model grows with said level, being firstly a mere impressed power, then piping and exchangers with prescribed water inlet conditions, then the complete circuit. Given the scope of this work, the matter is discussed in the next section, based on a representative example that synthetically covers all the detail levels.

4 Application

This section illustrates how, along the proposed approach, scalable-detail models are able to support a designer through the phases of a typical project. For simplicity, the addressed design refers to the temperature control of a single room. The room is $3 \times 3 \times 2.5$ m in size, surrounded by walls of 0.4 m thickness. Concerning the walls, their thermal conductivity is 1.91 W/(mK) , their density is 2400 kg/m^3 and their thermal capacity is 880 J/(kgK) . The convective heat transfer coefficient between the walls and the air of the room is $5 \text{ W/(m}^2\text{K)}$, while that between walls and the environment is $10 \text{ W/(m}^2\text{K)}$. The temperature of the environment that surrounds the room is kept constant at $10 \text{ }^\circ\text{C}$. The design objective is to maintain the air temperature in the room at $20 \text{ }^\circ\text{C}$.

4.1 Level 0: overall static energy needs assessment

In this phase, the designer's question is "how much power is needed in order to maintain the room (or a building) at a certain temperature level, given the envelope transmittance and assigned environmental conditions?" The answer to this (level 0) question can be obtained by static models such as that of figure 1.

At this level, transients are neglected, and heat flow rates are computed based only on thermal conduction and convection at steady state, when the temperature of the room has reached the desired value.

There is not the space here to enter into Modelica details. Suffice however to say that in figure (1) the air model (white cube) is a mere heat capacity, the wall models are multilayer thermal resistances plus an additional heat capacity, that when evaluated as parameter causes the Fourier-based heat transfer law to

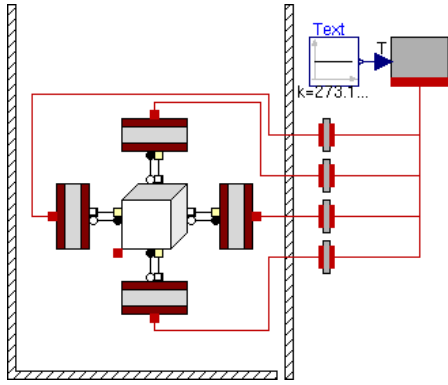


Figure 1: (Level 0) static analysis of the room only, considered as a mass of air at constant temperature $T = 20^{\circ}\text{C}$.

switch from dynamic to static in the case of zero capacity. Walls are connected to the air with two connectors: one simply carries the temperature as effort and the heat rate as flow variable, while the second conveys information about the air velocity for the subzonal room model mentioned later on: needless to say, such information is not used at the detail level of this section. The “T” block on the upper right side simply prescribes the temperature on its temperature/heat rate connector.

4.2 Level 1: dynamic energy needs assessment and local controls

According to the static model of figure (1), the power needed to maintain this steady state condition is 647.79 W . Scaling up the level of detail, this first result can be compared with a dynamic simulation.

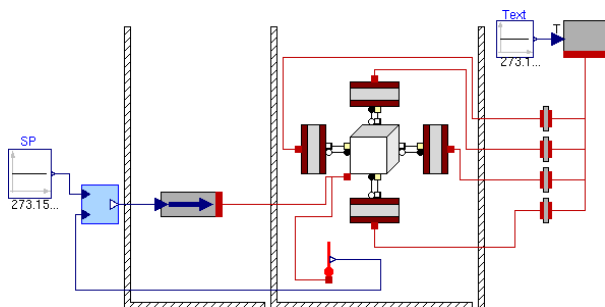


Figure 2: (Level 1) dynamic analysis of the room only, plus local controls. At this stage walls and air within the room are dynamic models. A simple control system, that directly injects power in the room, is introduced.

At this level, see figure (2), heat storages are considered, therefore heat capacity of walls and air are

included. A very simple control system is also introduced at the local level, while the conditions of the heating water centrally supplied are still impressed. The presented analysis is therefore *de facto* a level 1 one, and local controls are represented: the case where controls are conversely idealised is here skipped for brevity. The Modelica elements of figure (2) are the same as those of figure (1), plus a block prescribing the heat rate on its connector (near the centre) and an antiwindup, continuous-time PI controller (on the left).

Figure (4) shows the temperature transient, while figure (5) reports the power supplied by the control system to the room in order to maintain the prescribed temperature. This analysis shows that at steady state the amount of power predicted by the static analysis was correct, and the peak of power asked to the heating system in order to satisfy a certain response is higher than the final value (about 745 W). It is clear that this analysis is more complete than the previous one, because without considering dynamic effects (i.e., sizing the equipment based on information provided by static models only) the risk of incorrectly estimating the real needs is notoriously high.

4.3 Level 2: the energy system is brought in

At this point the question is “How does the energy (heating) system need to be sized and controlled in order to provide the required power to the system?” Such a question can be answered by further detailing the model as indicated before, but of the focus is set on the energy system exclusively, one could detail that system and at the same time scale down the level of complexity of the room, for example re-considering it as a mass of air at constant temperature (the worst case is when the temperature of the room has reached its maximum, i.e., the Set Point value of 20°C).

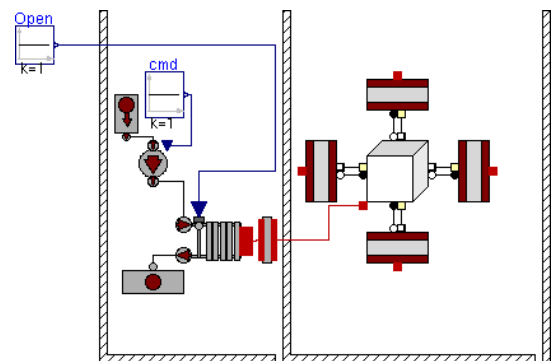


Figure 3: Level 2 (simplified) analysis of the heater only.

Figure (3) shows the new scheme. As anticipated

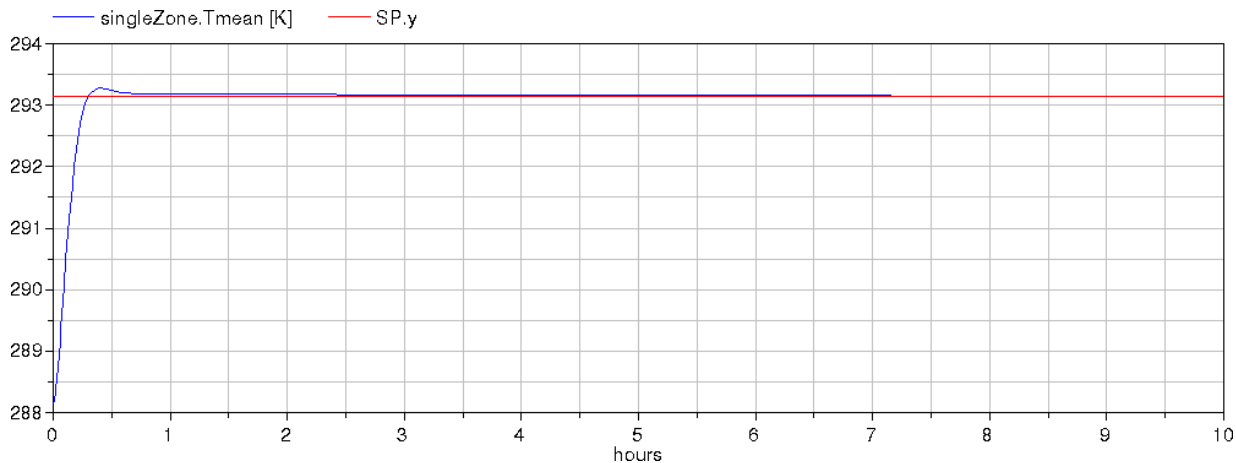


Figure 4: (Level 1) temperature of the air within the room (K).

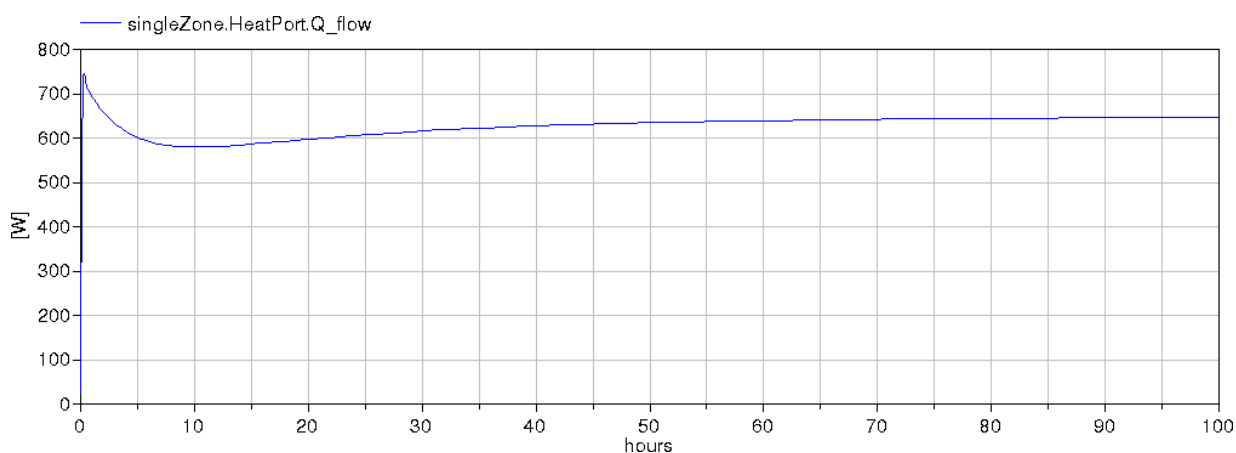


Figure 5: (Level 1) power supplied to the room (W).

the accent is posed on the heating system, that is not merely considered as an ideal heat flux injected in the room, as it was before. Given a certain quantity of hot water (assumed to be at 75°C) coming into the heater, its characteristics are investigated in order to release a certain power to the air.

In figure (3), the heating system is represented by a lumped-parameter model of the heater (an exchanging tube plus a metal mass), a pump described by a head/flow characteristic, and a source and a sink node prescribing respectively the heating fluid pressure and temperature, and the discharge pressure. Connectors allow for compatibility with lower-detail models, apparently.

Of course the so obtained results need checking against the full level-2 model, which is however omitted here for brevity. Notice however how the level of detail can be scaled in a non-uniform way throughout the model: the proposed level definition is therefore just a guideline, that the flexibility of the object-oriented approach allows the analyst to tailor accord-

ing to the particular question needing an answer.

4.4 Level 3: complete model

After sizing the main components that compose the system, the overall (level 3) model can be set up and simulated. At this level (fig. 6) both the dynamics of the room and of the heating system are taken into account, and also the central controls are represented. In fact, as can be seen, the heating system now includes a model of the boiler, accounting for the water heat balance and having as input the fuel flow rate, while the combustion process is not described and simply replaced by a fixed power released to the water and computed as the fuel flow times its heating value. Optionally a static efficiency curve can be introduced, which is however a useless detail in the context of this work.

The purpose of this analysis is the tuning of the control system. As a consequence, at this level the designer can verify that the sizing decisions previously taken are correct (and if not go back and size again).

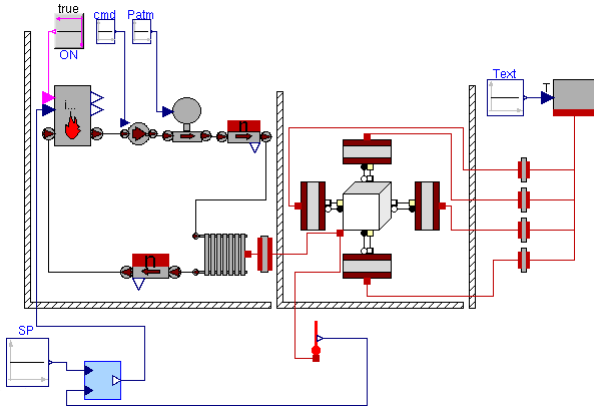


Figure 6: (Level 3) dynamic analysis of the room together with the heating system.

The tuning of the control system is done on a simple but reliable model that reflects all the dynamics that are part of the system (heating system, air and walls). The controller defined at this level may be used in more detailed descriptions.

4.5 More detail when needed

To further show the flexibility of the proposed *modus operandi*, one may need to reach even deeper levels of detail with respect to the main ones envisaged above. For example, up to now, the air within the room has always been treated as a unique entity (zero-dimensional model) and thus it had the same temperature, pressure, and so on, in every point. If necessary, the proposed model structuring allows to introduce more realistic approximations, for example based on a grid of sub-volumes.

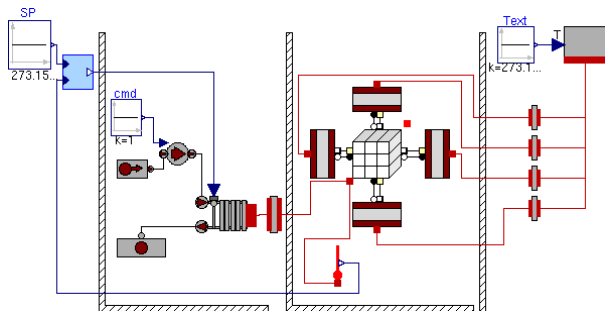


Figure 7: Dynamic analysis of the room together with the heating system. In this case the room is not considered as a single volume but is split into a coarse grid of sub-volumes.

With such a model it is possible to describe the motion of the air within the room, and more important, the temperature distribution within it. So, having a (more) detailed description of the temperature distribution of

the air contained in the room, problems like that of positioning the heater and the sensor in different places may be tackled. Notice that in figure (7) the heating system is described at an intermediate level (heater but no boiler); of course any variation is possible.

Results in figure (8) and (9) evidence how positioning the sensor in different places may vary the behaviour of the overall system. In particular, in the first case, the sensor is positioned on the left wall (the one where the heater too is placed), while in the second one, the sensor is on the opposite wall. In the first case the temperature is clearly underestimated, while in the other one it is overestimated, with the apparent consequences on transients and consumption. Simulating for 24 hours, in fact, the energy consumed in the models is 15.31 kWh against 13.85 kWh with a difference of 1.46 kWh. Such differences, when computing the overall consumption of a building over a year, may be significant.

On a similar front, one may want to describe in greater details (preserving the interface, of course) the energetically active components. To show an example, we report the model of the boiler in figure 6.

```

model Boiler_scalableDetail
  parameter Time Tc1=5 "Closed loop time constant for ideal control";
  parameter Time Thc=20 "Free cooling time constant for ideal control";
  parameter Real Kpi=1 "PI gain";
  parameter Time Tipi=10 "PI integral time";
  parameter Power Phmax=30000 "Max heating power";
  parameter Volume V=0.1 "Volume";
  parameter SpecificHeatCapacity cp=4186 "Heating fluid cp";
  parameter Density ro=1000 "Heating fluid density";
  parameter CelsiusTemperature Tstart=25 "Initial fluid temp";
  parameter Real eta0=0.6 "Min efficiency";
  parameter Real eta1=0.9 "mMx efficiency";
  parameter Real HH=48e6 "Fuel LHV";
  parameter Real Nm3_kg = 1.3942 "Nm3/kg ratio (default methane)";
  parameter Integer detailLevel=0 "Detail level";
  Modelica.Blocks.Interfaces.RealInput To;
  Interfaces.pwTinlet inlet; // connectors with pressure and flowrate
  Interfaces.pwTOutlet outlet;
  Modelica.Blocks.Interfaces.BooleanInput ON;
  Modelica.Blocks.Interfaces.RealOutput Pc;
  Modelica.Blocks.Interfaces.RealOutput Ec;
  Real Ph; // Heating power
  Real eta; // Efficiency
  Real wc; // Fuel flowrate
  Real Nm3tot(start=0); // Accumulated fuel consumption in Nm3
protected
  CelsiusTemperature Tfoc(start=Tstart) "Outlet temp, ideal ctrl";
  Real PIfb(start=0) "Internal signal for PI antiwindup";
equation
  inlet.p = outlet.p;
  inlet.w+outlet.w = 0;
  cp*ro*V*der(outlet.T) = inlet.w*cp*inlet.T+outlet.w*cp*outlet.T+Ph;
  eta = noEvent(max(eta0,
    min(eta1,eta0+(eta1-eta0)*Ph/Phmax)));
  Pc = Ph/eta;
  Pc = wc*HH;
  der(Ec) = Pc;
  der(Nm3tot) = wc*Nm3_kg;
  if detailLevel==0 then
    // Detail 0: ideal control of outlet temperature
    // NOTE: this implies no Ph saturation, hence Ph may become negative;
    // use this detail only for very first-cut studies and beware if
    // energy use estimates are involved
    outlet.T = Tfoc;
    PIfb = 0;
    if ON then
      Tfoc+Tc1*der(Tfoc) = To;
    else
      Tfoc+Thc*der(Tfoc) = inlet.T;
    end if;
  elseif detailLevel==1 then
    // Detail 1: PI control of outlet temp, no high Ph saturation (low only)
    Tfoc = Tstart;
    if ON then
    
```

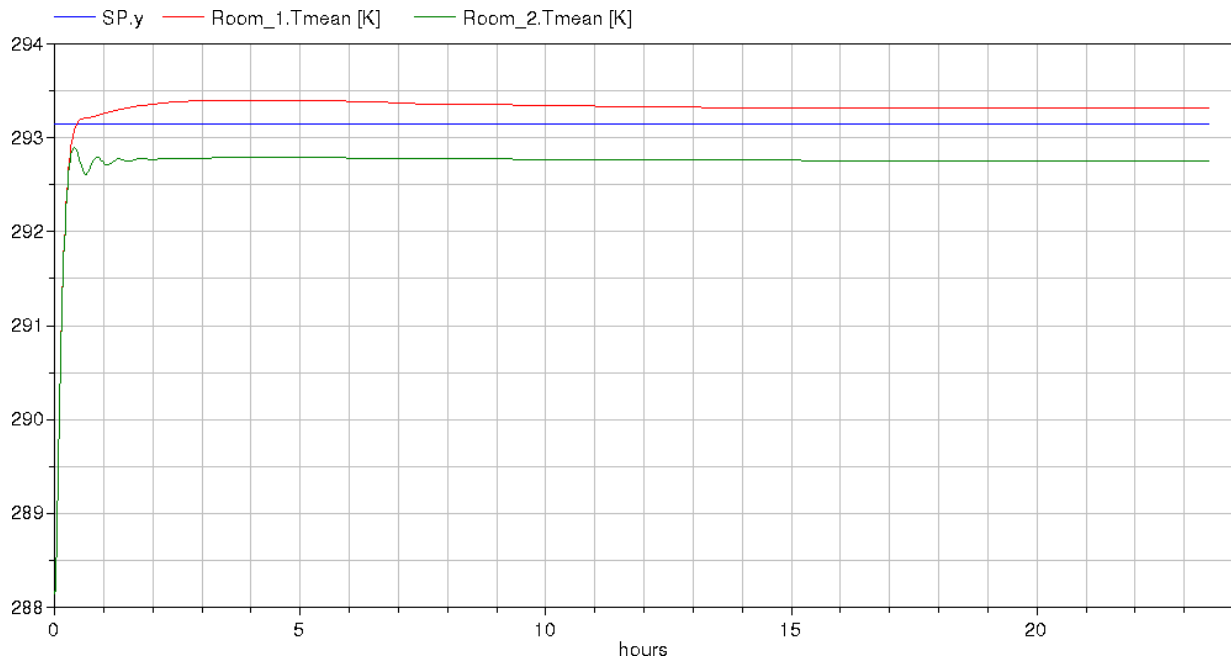


Figure 8: Measured (controlled) room temperature with different sensor positions (K).

```

    Ph = noEvent(max(0.0,Kpi*(To-outlet.T)+PIfb));
    Ph = PIfb+Tipi*der(PIfb);
  else
    Ph = 0;
    PIfb+0.01*Tipi*der(PIfb) = 0; // quick reset
  end if;
elseif detailLevel==2 then
// Detail 2: PI control of outlet temp, both high and low Ph saturation
Tfoic = Tstart;
if ON then
  Ph = noEvent(min(Phmax,max(0.0,Kpi*(To-outlet.T)+PIfb)));
  Ph = PIfb+Tipi*der(PIfb);
else
  Ph = 0;
  PIfb+0.01*Tipi*der(PIfb) = 0; // quick reset
end if;
end if;
// ...further levels of detail are clearly possible (combustion,...)
end Boiler_scalableDetail;

```

Notice how the same model can be used for sizing the equipment and assessing the central controls, consistently with the proposed way of using the simulation tool along the evolution of a project. Also, the use of convenient top-level parameters allows to use a single model, tailoring the detail level of its parts as needed. Of course the same result could have been obtained by exploiting model replaceability, but in the opinion of the authors, keeping all of a component's behaviour within a single model enhances readability (although of course the matter is largely subjective).

5 Conclusions

The use of object-oriented models *throughout* a project relative to building energy efficiency was discussed. Based on the authors' experience, one major weakness of most approaches to date is the lack of a library structuring conceived so as to follow the nec-

essary modifications of the required detail level, in all or part of the model, with the maximum ease of use on the part of the designer.

Along such a reasoning, a library structuring was proposed, and preliminarily demonstrated by applying it to quite simple case, yet complete enough to be a representative example. According to such initial results, it appears that a research effort specifically aimed at an effective library structuring as perceived by the user when managing models along a project, can be very beneficial for a better acceptance of the object-oriented paradigm, and a better exploitation of its possibilities.

Ongoing research is on the realisation of a complete library based on the envisaged structuring, both integrating the available wealth of literature results, and introducing *ad hoc* models simplifications, especially in a view to easing the task of aligning models of different detail levels with the minimum effort.

References

- [1] F. Felgner, S. Agustina, R. Caldera Bohigas, R. Merz, and L. Litz. Simulation of thermal building behaviour in Modelica. Oberpfaffenhofen, Germany, 2002.
- [2] F. Felgner, R. Merz, and L. Litz. Modular modelling of thermal building behaviour using Mod-

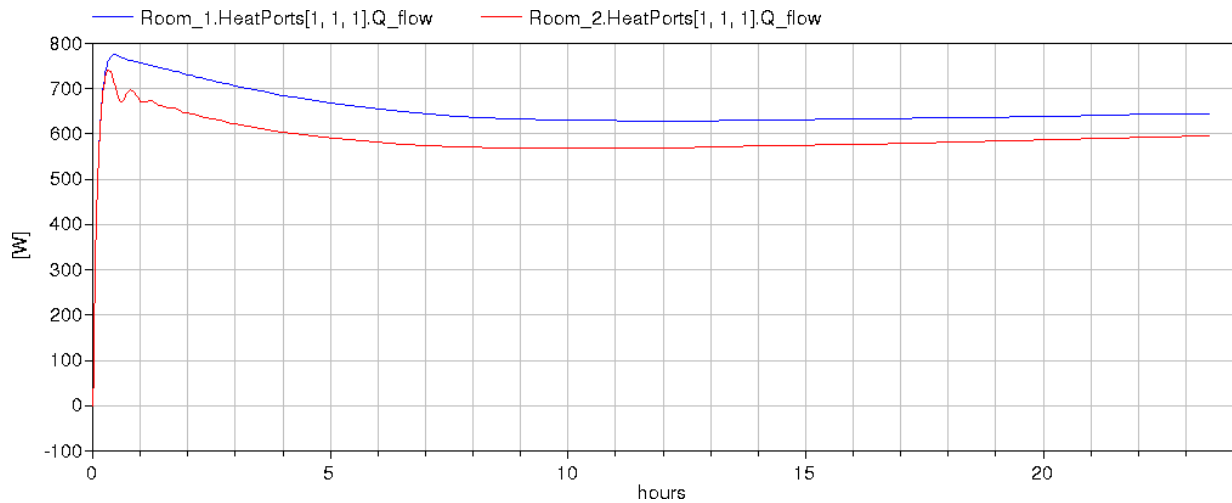


Figure 9: Power consumption with different sensor positions (W).

elica. *Mathematical and computer modelling of dynamical systems*, 12(1):35–49, 2006.

- [3] M. Janak. Coupling building energy and lighting simulation. Kyoto, Japan, 2000.
- [4] M. Wetter. Modelica-based modeling and simulation to support research and development in building energy and control systems. *Journal of Building Performance Simulation*, 2(1):143–161, 2009.
- [5] M. Wetter. Modelica library for building heating, ventilation and air-conditioning systems. Como, Italy, 2009.

Real-Time Simulation of Vapour Compression Cycles

Christian Schulze
TLK-Thermo GmbH,
Hans-Sommer-Str. 5, 38106 Braunschweig, Germany
C.Schulze@TLK-Thermo.de

Manuel Gräber
Technische Universität Braunschweig, Institut für Thermodynamik
Hans-Sommer-Str. 5, 38106 Braunschweig, Germany

Michaela Huhn
Technische Universität Clausthal, Institut für Informatik
Julius-Albert-Str. 4, 38678 Clausthal-Zellerfeld, Germany

Uwe Grätz
ITI GmbH
Webergasse 1, 01067 Dresden, Germany

Abstract

This paper shows a tool chain of a set of ready-to-use tools and libraries that enables the dynamic real-time simulation of vapour compression cycles. A new approach for calculation of fluid properties and numeric efficient component models are applied. As an Hardware in the Loop application a vapour compression cycle is exported to Scale-RT [5] using SimulationX [11] and connected to a hardware PI-Controller in order to realize a superheating control.

Keywords: Real-Time Simulation, Vapour Compression Cycle, Tool Chain

1 Introduction

Whereas so far simulation aimed for conceptual validation in the early concept phase, nowadays we find an increasing need for real-time simulation or even real-time execution of models on micro-controllers. For example Hardware in the Loop (HiL) is an important technique for testing hardware controllers in a simulated environment. It enables the evaluation of a controller on very unlikely test cases which can be implemented easily using simulation, which applies also for vapour compression cycles.

Model-based controllers for a vapour compression cycle can be developed on the basis of a Modelica model. Take the following approaches: A simulation executed on those controllers may on the one hand be used to replace some sensor signals by simulation results; on the other hand the failure of a sensor may be detected by comparing the sensor signals with the simulation results. Furthermore nonlinear model predictive control can be developed based on those Modelica models.

The numeric efficiency of the models and the fluid property calculation methods limits the complexity of the whole model due to the limited calculation speed of the CPU. In case accurate system modelling is needed the numeric efficiency of the component models and the fluid property calculation methods should be improved.

For the application on vapour compression cycles until now there is no numeric efficient ready-to-use tool chain presented that enables efficient simulation of those systems under varying circumstances. In this paper we present a part of the model library and the tool chain developed by the 4 authors of this paper.

The presented tool chain bases on a model library of thermal components, a fluid property library, various simulators and hardware environments as well as a profiling method.

This tool chain and the libraries are exemplified

on a R-407C heat pump cycle with a hardware PI-controller as superheating control, using SimulationX as a compiler and Scale-RT as hardware.

2 Real-Time Fluid Property Library

Profiling results of various thermodynamic systems show that the calculation of fluid properties has a major impact on the model runtime, so the calculation methods for fluid properties must be reconsidered. These results have been gained from the profiling method of the presented tool chain [20]. This method is briefly described in section 4.

For modelling of thermodynamic systems it is usually necessary to have access to the properties of the used media. There is a large number of methods to calculate fluid properties starting from the perfect gas theory to the fundamental equations of state. Those methods differ in the underlying theory, calculation speed, precision, amount of data needed and internal consistency.

For many dynamic simulations of vapour compression cycles a very high precision of the fluid properties is needed to describe the correct pressure levels as well as the correct temperature curve over the heat exchanger length, especially when modelling a superheating control. E.g. the enthalpy of evaporation and the vapour pressure curve have a high influence on those results but they are drawn from the whole fluids property description. Due to this reason fundamental equations of state are employed frequently for simulation of thermodynamic systems like these.

The Helmholtz Potential can be calculated from a fundamental equation of state as a function of density and temperature what from every thermodynamic state variable can be drawn from it. As physical processes often are described by enthalpy differences and pressure differences, the calculated state points frequently are given by pressure and specific enthalpy.

Using a fundamental equation of state the calculation of fluid properties at a given pressure and specific enthalpy requires a numerical solving process. As a result of this, simulations based on fundamental equations of state require a varying CPU work load at a high level and should be avoided for real-time application.

This paper different approach for calculation of fluid properties: instead of Modelica the calculation

methods are implemented in C. This way the the solving methods and solver parameters may be adapted suitable for this application therefore performance is gained. As depicted in figure 7 the fluid property calculation source code is added to the model source code after the export from SimulationX before compiling.

In order to decrease the CPU work load caused by the fluid property calculation other methods instead of fundamental equations of state can be used. In the following we concentrate on calculation methods that are not based on an equation of state.

Fluid properties can be tabulated and interpolated linearly on the basis of equation (1). This way the CPU work load can be reduced significantly but the precision of the results between two tabulated grid points is poor due to the linear approximation. As a countermeasure the distance between two tabulated grid points must be reduced to a minimum so the amount of data is very high. Either the thermodynamic properties or the results and derivatives of a thermodynamic potential [1] can be tabulated.

$$T(p, h) = \sum_{i=0}^1 \sum_{j=0}^1 a_{ij} p^i h^j \quad (1)$$

In addition to the precision of a single value the internal consistency of linearly interpolated data between different values is poor, too. For example if temperature and the specific heat capacity are tabulated then the specific heat capacity calculated from the difference quotient of two temperatures differs from the linear interpolated heat capacity. So $\int c_p dT$ does not approach Δh and hence the energy balance may be violated depending on the model equations.

Spline interpolation is another way to close the gap between to tabulated grid points. Spline interpolation requires the tabulation of the spline function coefficients (see equation (2)). These coefficients can be calculated from the tabulated grid points under the constraint that the results during transition from one set of coefficients to another set of coefficients must be continuously differentiable. This technique is discussed and exemplified for the properties of Water by Kunick [12].

$$T(p, h) = \sum_{i=0}^3 \sum_{j=0}^3 a_{ij} p^i h^j \quad (2)$$

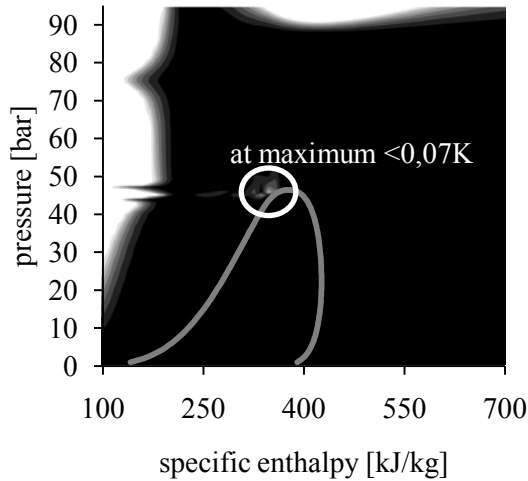


Figure 1: Deviation of temperature compared with Refprop for R-407C PPF.

This paper uses another approach for calculation of one and two phase fluid properties. The basic idea is a functional fit for the enthalpy dependency at various pressure levels in combination with a linear interpolation between those fit functions [21]. This may be interpreted as an array of curves, whereas the gaps between two curves are closed via linear interpolation. Figure 1 shows the error of the temperature calculated from this method compared to the results from Refprop [15] R-407C Pseudo Pure Fluid [14].

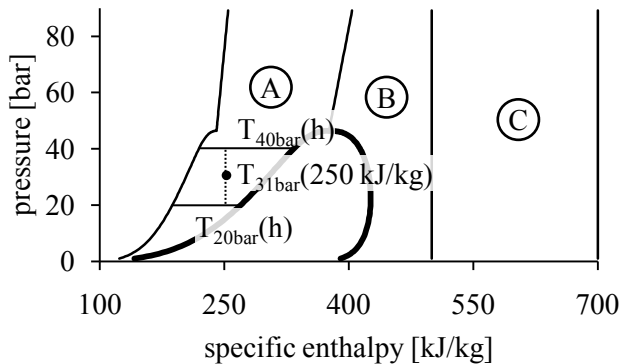


Figure 2: A fluid property is interpolated linearly between the fit equation at upper and lower pressure level, using one set of coefficients for each region A, B and C per pressure level.

Figure 2 illustrates the method of calculating fluid properties using equation (3): The coefficients for the fit equation are tabled for about 200 pressure levels from the triple point up to 85 bars, but the number of pressure levels will be reduced in future. A separate set of coefficients is needed for each region A, B and C. The outstanding fea-

ture of this method compared to the simple linear interpolation is its significantly improved consistency and the reduction of required data.

$$T(p, h) = T_{p_i}(h) + \frac{T_{p_j}(h) - T_{p_i}(h)}{p_j - p_i}(p - p_i) \quad (3)$$

3 Component Library for Real-Time Applications

TLK-Thermo GmbH and the Institut für Thermodynamik of TU Braunschweig develop and maintain the Modelica library TIL [19, 8]. This is a component library for modelling complex thermodynamic systems such as heat-pumps, air conditioning and refrigeration cycles as well as organic rankine cycles, and TIL has been used in various academic and industrial research projects. Combined with the real-time fluid property library described in section 2 a lot of systems modelled with TIL can already be run on real-time hardware. Of course, model complexity has to be adapted to the specific needs. For example complex pressure drop and heat transfer correlations will lead to too large computational effort. In this section we will shortly introduce TIL and give an outlook on further model development for real-time applications.

The distinction of different fluid types is a basic design concept of TIL. E.g. there are three valves, one for incompressible liquids, one for ideal gases and another one for refrigerants. Each model can be adapted to the specific physical behaviour. The fluid properties of in incompressible liquid only depend on the temperature, so these models are simpler compared to those of the refrigerant.

3.1 Heat Exchangers

Figure 3 shows the four different heat exchanger modelling approaches implemented in TIL. Whereas the finite volume and moving boundary approach are designed for dynamic simulation, the NTU-Method and finite differences approach simplify the dynamic behavior of the system.

Heat exchanger models based on finite volumes can give a good picture of the real physical behaviour inside. The more cells are used to describe

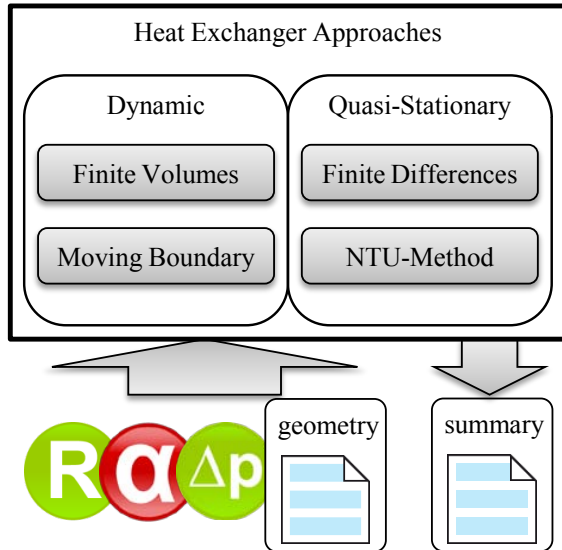


Figure 3: Four Heat Exchanger modelling Approaches implemented in TIL sharing the same interface for fluid properties, pressure drop, heat transfer, geometry and summaries.

the heat exchanger the more detailed will this picture will be. Of course a high number of cells causes a high CPU work load. The general structure of the implemented heat exchanger is shown in figure 4.

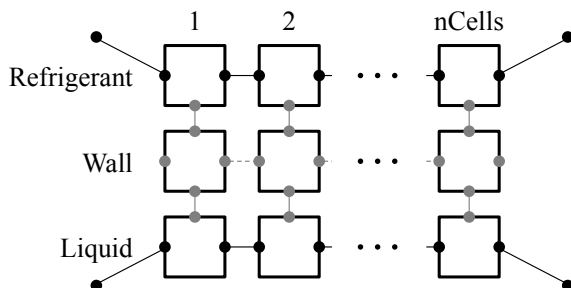


Figure 4: Finite Volume Tube and Tube Heat Exchanger Model based on cell models

In case of an Tube And Tube Heat Exchanger the wall, the refrigerant and the liquid are discretized one dimensionally into the same number of cells as depicted in figure 4. As a simplification the pressure change $\frac{dp}{dt}$ in the cells at the same pressure level is set equal as described by Lemke [13]. As a result of this there is only one continuous time state for the pressure at one pressure level and the CPU work load is reduced significantly.

For dynamic simulation of heat exchangers the moving boundary approach is the second ap-

proach. The Heat Exchanger is lumped into 3 cells, a cell covering the superheated region, one covering the two phase region and another one covering the subcooled region [9]. The lengths of those cells change dynamically, so a suitable heat transfer correlation can be implemented for each section. The precision increases significantly compared to finite volumes with a small number of cells.

The third approach for very fast stationary simulation of heat exchangers is based on the NTU-Method (Number of Transfer Units) [10]. Assuming that the transient effects of heat and mass transfer are negligible the resulting equation system is very small and can be solved quickly. In many cases a stationary model of a vapour compression cycle enables sufficient description of the system behaviour.

The fourth heat exchanger modelling approach is based on the finite difference method. In this approach derivatives of differential equations for temperatures and mass fractions are approximated using finite difference equations. The model is optimized for fast steady state solution at high spatial resolutions and detailed modelling of heat and mass transfer processes [22].

3.2 Efficiency based Compressor

In the compressor model used for the case study of this paper the outlet state is calculated from the number of rotations per second, the displacement, the volumetric and the isentropic efficiency.

$$\dot{V} = \rho_{in} \cdot n \cdot displacement \cdot \eta_{vol} \quad (4)$$

$$h_{out} = h_{in} + \frac{h_{isen} - h_{in}}{\eta_{isen}} \quad (5)$$

3.3 Valve

The Valve is based on Bernoulli's equation. The transition from positive to negative mass flow is smoothed.

$$\dot{m} = \sqrt{2\rho_{in} (p_{in} - p_{out})} \quad (6)$$

3.4 Filling Station

The filling station is a component to control the total mass of refrigerant inside the vapour compression cycle. The mass inside the cycle cannot

be set directly at initialization but this component enables the correction of the mass during simulation.

3.5 Ideal Separator

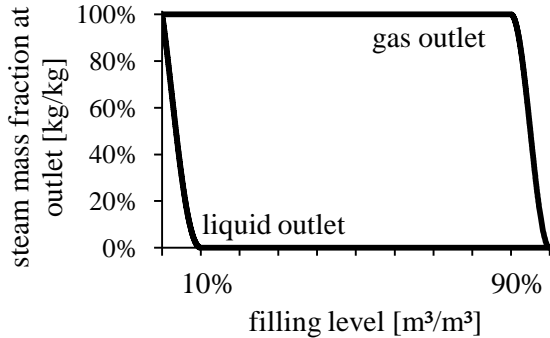


Figure 5: Ideal Separator Characteristic

The ideal separator is a component with a preset volume having 2 outlets and is used to separate liquid and vapour, the characteristic is illustrated in figure 5. In case of a filling level greater than 90% liquid refrigerant will come out of the gas outlet. Between 10% and 90% filling level the separation of liquid and vapour is done perfectly. In case of a filling level below 10% gaseous refrigerant will come out of the liquid outlet. The transition at 10% and 90% filling level is smoothed and hence continuously differentiable.

4 Tool Chain

Many real-time platforms are available to support HiL and RCP such as dSPACE [6], RT-LAB/QNX [17], NI-Veristand [16] or Scale-RT [5]. Scale-RT is a Real Time Operating System based on Linux using the Xenomai Kernel extension running on common desktop PCs. For interaction with the real world, IO-Hardware has to be installed. The RT-Environment used in the case study is a common Desktop PC with a Intel Core2Duo E8400 @3GHz, 4 GB RAM, Scale-RT 5.1.2 and two NI PCIe-6259 as IO-Interface.

Usually Real Time Operating Systems execute a real-time application with a higher priority. The processing of hardware events and the execution of non real-time applications is delayed [7].

Simple IO-Block based controller models can i.e. be created in tools like Matlab Simulink or the

open source project SciCos [3]. Those models can easily be exported to real-time targets. To export Modelica models to a real-time target including external C-Source this external source code has to be added to exported source code before compiling it.

Additional blocks or interfaces have to be implemented into the simulation environment as interfaces to the IO-Hardware. To access the IO-Hardware those blocks have to be instanced and configured in the model.

Depending on the Real Time Operating System (RTOS) tools like Matlab Simulink or SimulationX may either have to offer a hardware driver for each hardware IO-Interface card or can use an internal driver framework of the RTOS. Whereas NI Veristand provides a simulation and hardware framework Scale-RT version 4.x does not provide such a framework. Since version 5.x of Scale-RT a new model framework has been implemented.

SimulationX 3.4 provides access to several IO hardware interfaces for Scale-RT 4.x but unfortunately it did not support the NI PCIe-6259 multi-IO driver for the comedi driver interface [4] yet. This interface has now been added to SimulationX and will soon be available.

On real-time targets usually a fixed step solver is used to integrate the differential algebraic equation system of the whole model. If the calculation time needed exceeds the fixed solver step size an overrun is caused. Depending on the Real Time Operating System this may terminate the simulation. The algebraic loops are solved using a modified Newton solver.

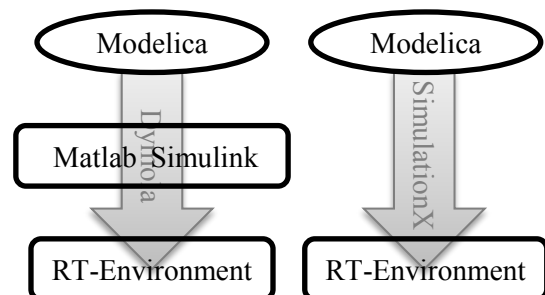


Figure 6: SimulationX enables direct Export to RT-Environment.

In contrast to Dymola SimulationX enables the direct export to a real-time environment in addition to the export via Matlab Simulink / Real Time Workshop as illustrated in Figure 6. The Real Time Workshop is used by many tools as

export interface to real time environments. Details on the export of models from SimulationX to Scale-RT and the optimization performed on it are described in [2].

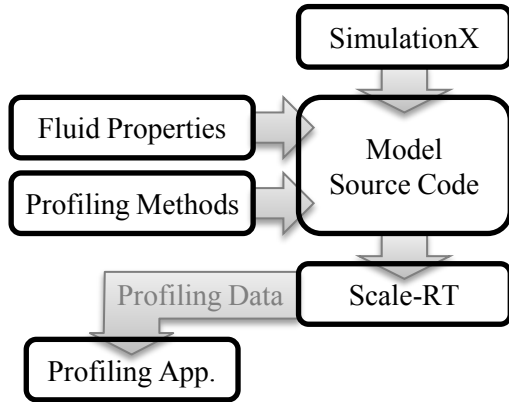


Figure 7: The instrumented model sends profiling data to a secondary application which saves that data

A Profiling method as presented in [20] has been added to this tool chain as well. Profiling of thermodynamic models on the real-time target itself reveals possible weaknesses of the exported model. The process of instrumentation and data saving for the profiling methods is depicted in figure 7. The source code of the exported model has to be modified, on the Real Time Operating System the time-consuming task of saving the data to the hard disk is done by a secondary non real-time application. This profiling method enables a general analysis of the DAE System.

5 Case Study: PI-Controlled R-407C-Cycle

In this section an exemplary application of the above discussed research results is described. We look at a R-407C water to water heat pump cycle. An electric expansion valve and a PI-controller are used to control superheating at evaporator outlet. Real-time simulation experiments are used to tune control parameters.

5.1 Model description

As starting point a model of the vapour compression cycle is built up using the libraries described in sections 2 and 3.

Although finite volumes cause a high CPU workload we decided to use heat exchangers based of 7

finite volumes each, in contrast to Pitchaikani et al. [18] using just one cell. The pressure drop in the heat exchangers is assumed to be negligible.

The model's graphical representation is shown in Figure 8.

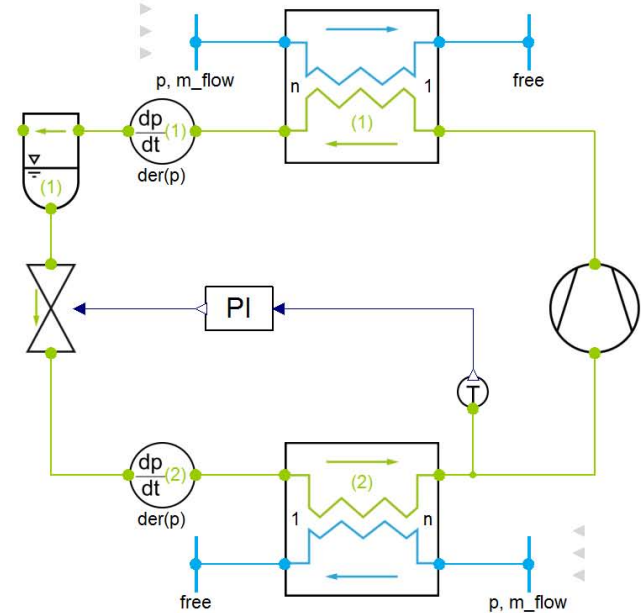


Figure 8: Diagramm of heat pump Modelica model.

Basically the cycle consists of compressor, controlled expansion valve, separator, condenser and evaporator. Superheating is measured at evaporator refrigerant outlet and transmitted to a PI-controller, which computes the expansion valve's opening, according to

$$u = K \left((y - y_s) + \frac{1}{\tau} \int y - y_s dt \right), \quad (7)$$

where u is the relative valve opening, and $y - y_s$ denotes the difference of measured superheating to its set point. Proportional gain K and integral time τ are constant parameters. Suitable values for these parameters are obtained by simulation experiments on a normal PC using the Modelica tool SimulationX. The next step is to use a real hardware controller and connect it to real-time simulated cycle.

5.2 Real-time experiment

The vapour compression cycle model runs on a Scale-RT real-time computer system. A hardware PI-controller is connected via I/O-Boards. Superheating and valve opening are transformed to ana-

integration step size	1ms
sample rate	10ms
number of continuous time states	60
number of integer variables	509
number of real variables	1811
number of external objects (refrigerant properties)	30
sizes of nonlinear systems of equations	2, 3, 9, 11, 3, 1, 9, 11
number of linear systems of equations (size 1)	120
model runtime on RT Environment	$\approx 2.5\text{ms}$

Table 1: Detailed Information about the exported Model

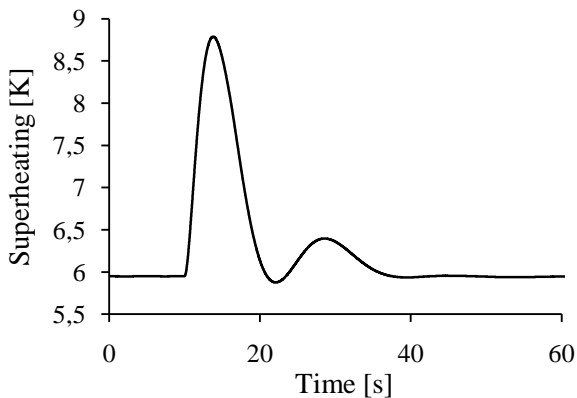


Figure 9: Superheating at evaporator refrigerant outlet.

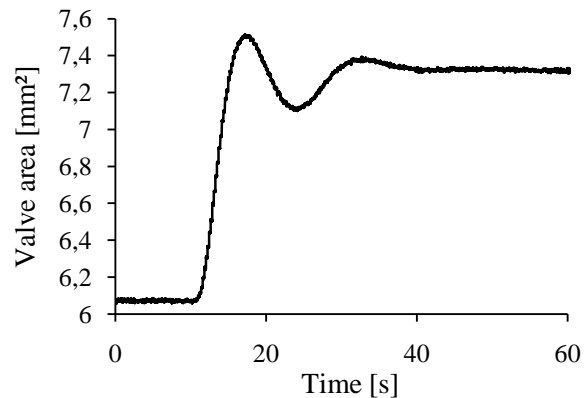


Figure 10: Opening area of expansion valve.

logue voltage signals. Now, the closed-loop performance can be tested under different boundary conditions.

Figure 9 and 10 show an example result. Starting from steady-state condition – superheating is close to its set point of 6K – a step of -5K is applied to the water temperature at evaporator inlet. The systems response is an immediate increase of the superheating and the controller reacts by opening the expansion valve. After one oscillation superheating setpoint is reached again.

6 Conclusion

In this paper a ready-to-use set of libraries as part of a tools chain for real time simulation applications is presented. The TIL Library as well as the TEMO fluid property library are used in this tool chain and exported to Real Time Operating Systems using SimulationX.

As one major contributor to the real time capability of the tool chain a new method for calculation of fluid properties of two-phase refrigerants

is developed and exemplified for R-407C, furthermore other interpolation based calculation methods are described. Four different implemented approaches for heat exchanger modelling – Finite Volumes, Moving Boundary, NTU-Method and Finite Differences – are presented.

Finally as one exemplary case study of the tool chain a numeric efficient and accurate modelling of a vapour compression cycle is presented.

7 Acknowledgement

Most part of this work has been funded by the German Federal Ministry of Education and Research (BMBF) in the project TEMO (grant 01|S08013C).

References

- [1] Trond Andresen. *Mathematical modelica of CO₂ based heat pumping systems*. PhD thesis, Norwegian University of Science and Technology, 2009.

- [2] Torsten Blochwitz and Thomas Beutlich. Real-time simulation of Modelica-based models. In *Proc. 7th Modelica Conference*, pages 386–392. The Modelica Association, 2009.
- [3] Roberto Bucher and Silvano Balemi. Scilab/Scicos and Linux RTAI - A unified approach. In *Proceedings of the IEEE Conference on Control Applications*, Toronto, Canada, August 2005.
- [4] COMEDI. Linux Control and Measurement Device Interface, 2011. URL <http://www.comedi.org>.
- [5] Cosateq GmbH & Co. KG. Scale-RT, 2010. URL <http://www.scale-rt.com/>.
- [6] dSPACE GmbH. dSPACE, 2011. URL <http://www.dSPACE.com/>.
- [7] Philippe Gerum. The XENOMAI Project - Implementing a RTOS emulation framework on GNU/Linux. Technical report, 2002.
- [8] M. Gräber, K. Kosowski, C. Richter, and W. Tegethoff. Modelling of heat pumps with an object-oriented model library for thermodynamic systems. *Mathematical and Computer Modelling of Dynamical Systems*, 16: 195–209, 2010.
- [9] M. Gräber, N. C. Strupp, and W. Tegethoff. Moving Boundary Heat Exchanger Model and Validation Procedure. In *EU-ROSIM Congress on Modelling and Simulation*, Prague, 2010.
- [10] F. P. Incropera, D. P. DeWitt, T. L. Bergman, and A. S. Lavine. *Fundamentals of Heat and Mass Transfer*. John Wiley & Sons US, 6th edition edition, 2006.
- [11] ITI GmbH. SimulationX, 2010. URL <http://www.simulationx.com>.
- [12] Matthias Kunick, Hans-Joachim Kretschmar, and Uwe Gampe. Fast Calculation of Thermodynamic Properties of Water and Steam in Process Modelling using Spline Interpolation. *Proceedings of the 15h International Conference on the Properties of Water and Steam*, 2008.
- [13] Nicholas C. Lemke. *Untersuchung zweistufiger Flüssigkeitskühler mit dem Kältemittel CO₂*. Number 73 in Forschungsberichte des Deutschen Kälte- und Klimatechnischen Vereins. Deutscher Kälte- und Klimatechnischer Verein, Holtzminden, 2005.
- [14] E. W. Lemmon. Pseudo-Pure Fluid Equation of State for the Refrigerant Blends R-410A, R-404A, R-507A, and R-407C. *International Journal of Thermophysics*, Vol. 24, No. 4, 24, 2003.
- [15] M. O. McLinden, S. A. Klein, E. W. Lemmon, and A. P. Peskin. *NIST thermodynamic and transport properties of refrigerants and refrigerant mixtures-REFPROP*. 2008.
- [16] National Instruments. NI VeriStand, 2011. URL <http://www.ni.com/veristand>.
- [17] OPAL-RT. RT-LAB, 2011. URL <http://www.opal-rt.com/product/rt-lab-professional>.
- [18] Anand Pitchaikani, Kingsly Jebakumar S, Shankar Venkataraman, and S. A. Sundaresan. Real-time Drive Cycle Simulation of Automotive Climate Control System. In *Proc. 7th Modelica Conference*, pages 839–846. The Modelica Association, 2009.
- [19] Christoph Richter. *Proposal of New Object-Oriented Equation-Based Model Libraries for Thermodynamic Systems*. PhD thesis, TU Braunschweig, 2008.
- [20] C. Schulze, M. Huhn, and M. Schüler. Profiling of Modelica Real-Time Models. In *Proceedings of the 3rd International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools*, volume 3, pages 23–32, 2010.
- [21] C. Schulze, W. Tegethoff, M. Huhn, and J. Köhler. Numerisch effiziente Berechnungsmethoden für die Stoffeigenschaften von Fluiden für die Systemsimulation. *DKV-Tagungsberichte*, 2010.
- [22] N. C. Strupp, R. M. Kossel, W. Tegethoff, and J. Köhler. Senkung des Kraftstoffverbrauches durch Optimierung der Leerlaufklimatisierung eines PKW mittels Hybridkühlung. In *DKV Tagung*, 2010.

Efficient hybrid simulation of autotuning PI controllers

Alberto Leva, Marco Bonvini*

Dipartimento di Elettronica e Informazione, Politecnico di Milano
Via Ponzio 34/5, 20133 Milano, Italy
{leva, bonvini}@elet.polimi.it

*PhD student at the Dipartimento di Elettronica e Informazione

Abstract

Autotuning methods are typically conceived as procedures, thus need simulating as digital blocks. However, when no autotuning is in progress, it is far more efficient to represent the tuned controller as a continuous-time system, to exploit variable-step integration. This manuscript presents the first *nucleus* of a Modelica library of autotuning controllers, where the problem just mentioned is tackled explicitly. The focus is here restricted to the PI structure, but the presented ideas are general.

Keywords: Autotuning; PI control; hybrid systems' simulation.

1 Introduction

It is universally acknowledged that PI and PID regulators significantly contribute to form the backbone of industrial controls [5, 3]. Also, in many applications and especially in recent years, their automatic tuning is of paramount importance for a quick system setup and an easy maintenance. As a result, an impressive quantity of autotuning rules can be found in the literature, see e.g. the vast review [17]; analogously, a large and steadily increasing number of industrial application and products are available, as testified by works such as [15].

Apparently, therefore, the simulation of PI(D) autotuners is a very interesting topic, for at least two reasons. From the standpoint of the analyst who performs system-level simulation studies, for example in a view to ease and speed the commissioning of a plant, autotuning is precious to reduce the time needed to parametrise the included regulators, that are often quite numerous. From the point of view of engineer who develops autotuning controllers, conversely, the possibility of testing a product (with a quasi-*replica* code representation) on realistic simulation models is

equally precious, since doing so allows to assess *a priori* its correct behaviour in the whole class of application it is intended for.

However, in a view to achieve efficient simulation, the presence of autotuning regulators poses a relevant issue. The problem is that autotuners are typically conceived as digital blocks, and for the sake of correctness and precision, so need to be their models. On the other hand, when no autotuning is in progress, the regulator behaves as a fixed-parameter dynamic system, thus it is far more convenient to represent it in the continuous-time domain, so as to exploit variable-step integration.

In such a context, this manuscript presents the first *nucleus* of a Modelica library of autotuning controllers, and concentrates on their hybrid representation, encompassing a continuous-time model of the controller, and a digital model of the autotuning part. After a brief theoretical review, a general structure for the necessary Modelica models is proposed as the main contribution, and an application that refers to a relay-based PI autotuner is presented. Simulation examples show the efficiency advantages of the presented hybrid representation with respect to a fully digital one.

2 Theoretical background

This work, although (as can be guessed) the proposed ideas are general, limits the scope to relay-based autotuning, and considers a one-degree-of-freedom PI written in the Laplace transform domain as the error-to-control transfer function

$$R(s) = K \left(1 + \frac{1}{sT_i} \right), \quad (1)$$

where K is the gain and T_i the integral time. The basic principle of relay-based autotuning was introduced in [1], and then developed in [2, 8, 4, 16, 9] and many other papers; a survey on the matter, for the interested reader, can be found in [21].

In extreme synthesis, the idea is to lead the controlled system to a limit cycle by substituting the controller to be tuned with a relay, as shown in figure 1.

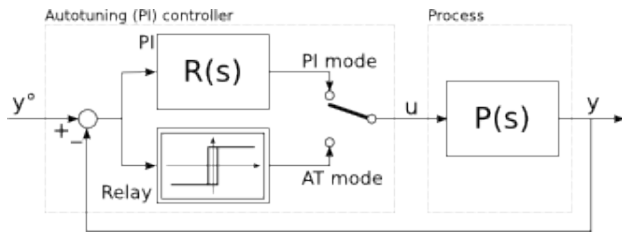


Figure 1: Basic scheme for relay-based (PI) autotuning.

Once said condition is established, by measuring the period and amplitude of the induced controlled variable's oscillation and by resorting to the well known describing function approximation, it is possible to estimate one point $\hat{P}(j\omega_{ox}) = P_{ox}e^{j\varphi_{ox}}$ of the process frequency response $P(j\omega)$, where ω_{ox} is the mentioned oscillation frequency. Then, to tune the PI, a point \bar{L} is chosen that the open-loop frequency response $L(j\omega) = R(j\omega)P(j\omega)$ has to contain, and the two parameters of the regulator $R(s)$ are found by solving the complex equation

$$R(j\omega_{ox})P_{ox}e^{j\varphi_{ox}} = \bar{L}. \quad (2)$$

A widely used specification in relay-based PI autotuning is the closed-loop phase margin φ_m , which is enforced in a straightforward way by forcing $L(j\omega)$ to cross the unit circle, at frequency ω_{ox} , in the point $\bar{L} = e^{j(\varphi_m - \pi)}$, with φ_m in radians.

In this work, a slight variant of the scheme shown in figure 1 is used, where the relay is hysteresis-free, or has so small a hysteresis to allow the real negative semiaxis to be considered its critical point locus, and there is an integrator cascaded to it. Doing so causes the oscillation to occur at the frequency where the phase of $P(j\omega)/(j\omega)$ is $-\pi$, i.e., that of $P(j\omega)$ is $-\pi/2$. The situation is illustrated in figure 2, where M denotes the frequency response magnitude of $P(s)/s$ at frequency $\omega - ox$

In this case, some computations omitted for brevity lead to determine the magnitude of $P(j\omega)$ at the oscillation frequency ω_{ox} as

$$P_{ox} = \frac{\pi^2 A}{8D}, \quad (3)$$

where A is the amplitude of the controlled variable's permanent oscillation, and D the relay swing. Selecting the process frequency response point with phase

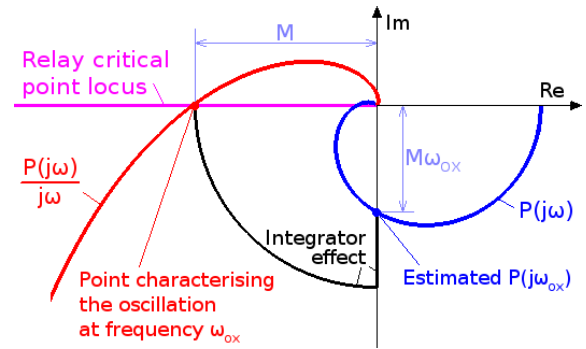


Figure 2: One-point identification with relay plus integrator feedback.

$-\pi/2$ is a convenient choice, since a PI regulator can only introduce a phase lag: the desired phase margin φ_m is in fact obtained by drawing from (2) the two real equations for magnitude and phase, whence the simple tuning rules

$$T_i = \frac{\tan(\varphi_m)}{\omega_{ox}}, \quad K = \frac{\tan(\varphi_m)}{P_{ox}\sqrt{1 + \tan^2(\varphi_m)}}, \quad (4)$$

that are used for the PI autotuner presented later on in this work.

Many variants of (4) exist in the literature, see e.g. [18, 20] or the so called “contextual autotuning” recently proposed in [12]. Moreover, the same tuning principle is applicable to the PID, and also to more complex regulator structure, possibly detecting and employing several points of $P(j\omega)$. The results shown here can be easily extended to any such case.

3 Modelica implementation

This section presents two Modelica realisations (the first fully digital, the second hybrid) of the considered autotuning methodology. In both cases, the icon of the resulting block is that of figure 3.

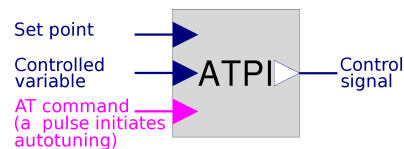


Figure 3: Modelica icon of the autotuning PI controller.

The block inputs are the set point and the process variable, plus a boolean one, a pulse on which initiates the autotuning procedure; the output is clearly the control signal. The initial values for K and T_i , as well as the required phase margin, are provided as parameters.

In both realisations, with reference to figure 1 and the relationships introduced above, the autotuning procedure is composed of the following steps:

1. start with the controller in PI mode;
2. when the AT pulse is received,
 - (a) initialise the relay plus integrator control,
 - (b) connect it to the process,
 - (c) and wait for a permanent oscillation;

in the quite simple procedure presented here, an oscillation is considered permanent when the difference between its period and that of the previous one is less than a percent defined as parameter, while - for the sake of safety in the face of possible outliers - a certain number of oscillations, defined as a parameter too, is counted unconditionally before proceeding;

3. when a permanent oscillation is detected, compute its frequency, and by means of (3) determine the corresponding process frequency response magnitude (the phase is $-\pi/2$);
4. apply (4) to tune the regulator, and finally switch back to PI mode.

It is worth noticing that any industrial realisation would be more articulated than those illustrated in the following. For example, some logic would need introducing to abort the procedure in the case of unexpected and/or possibly harmful system behaviours, a confirmation should be requested to the operator in order to accept or decline the proposed parameters prior to updating the PI, and so forth. Such features are however omitted here since they are lengthy to discuss in the necessary detail, and substantially inessential for the purpose of this work.

3.1 Fully digital version

Based on the procedure sketched above, it is quite simple to write a digital Modelica model like that reported below, together with some comments that should be explicative enough compatibly with space limitations.

```

model ATPIrelayNCdigital
import Modelica.Constants.*;
parameter Real K0 = 1 "Initial K";
parameter Real Ti0 = 10 "Initial Ti";
parameter Real slope = 0.1 "relay integrator gain (control slope)";
parameter Real permOxPerc = 5 "% diff to take oscillations as equal";
parameter Real pm = 45 "reqd phase margin in degrees";
parameter Real CSmax = 1 "upper control saturation value";
parameter Real CSmin = 0 "lower control saturation value";
parameter Integer nOxMin = 3 "oxs to wait for unconditionally";
parameter Real Ts = 0.1 "sampling time";
protected

```

```

discrete Boolean UP; // relay is in the up state
discrete Real lastToggleUp; // instant of last toggle to up
discrete Real period; // measured ox period
discrete Real wox; // measured ox frequency
discrete Real Pox; // measured process mag at wox
discrete Real rPVmax; // service variables to measure the
discrete Real rPVmin; // min and max values of the process
discrete Real rCSmax; // variable and the control during
discrete Real rCSmin; // oscillations
discrete Real K; // PI gain
discrete Real Ti; // PI integral time
discrete Real e; // error (Sp-PV)
discrete Real CSp; // proportional part of CS
discrete Real CSi; // integral part of CS
discrete Integer nOx; // ox counter
discrete Integer iMode; // mode: 0 is PI, 1 autotuning
Modelica.Blocks.Interfaces.RealInput SP;
Modelica.Blocks.Interfaces.RealInput PV;
Modelica.Blocks.Interfaces.RealOutput CS;
Modelica.Blocks.Interfaces.BooleanInput ATreq;
algorithm
when initial() then
iMode := 0;
K := K0;
Ti := Ti0;
end when;
when ATreq==true then
iMode:=1;
end when;
when sample(0,Ts) then
e := SP-PV;
if iMode==0 then // PI mode
CSp := K*e;
CSi := pre(CSi)+K*Ts/Ti*e;
CS := CSp+CSi;
if CS>CSmax then
CS:=CSmax;
end if;
if CS<CSmin then
CS:=CSmin;
end if;
CSi := CS-CSp;
end if;
if iMode==1 then // AT mode
if pre(iMode)==0 then // 1st step, initialise
wox := 0;
Pox := 0;
rPVmax := PV;
rPVmin := PV;
rCSmax := CS;
rCSmin := CS;
lastToggleUp := time;
nOx := 0;
end if;
if UP==false and PV<=SP then // Manage relay
UP := true;
end if;
if UP==true and PV>SP then
UP := false;
end if;
if UP==true then
CS := CS + slope*Ts;
else
CS := CS - slope*Ts;
end if;
if PV>rPVmax then // record max and min for PV and CS
rPVmax := PV;
end if;
if PV<rPVmin then
rPVmin := PV;
end if;
if CS>rCSmax then
rCSmax := CS;
end if;
if CS<rCSmin then
rCSmin := CS;
end if;
if UP==true and pre(UP)==false then // tune if perm ox
period := time-lastToggleUp;
lastToggleUp := time;
if period>0 and nOx>=nOxMin
and abs(period-pre(period))/period
< permOxPerc/100 then
iMode := 0;
wox := 2*pi/period;
Pox := pi^2*(rPVmax-rPVmin)/8/(rCSmax-rCSmin);
Ti := tan(pm/180*pi)/wox;
K := tan(pm/180*pi)/(Pox*sqrt(1+(tan(pm/180*pi))^2));
CSi := CS-K*Ts/Ti*e; // re-initialise the PI after AT
end if;
rPVmax := PV;
rPVmin := PV;
rCSmax := CS;
rCSmin := CS;
nOx := nOx+1;
end if;
end if;
end when;
end ATPIrelayNCdigital;

```

3.2 From fully digital to hybrid

When everything is digital, things are simple, and the only issue to care about is to correctly manage the regulator tracking while the relay is driving the control signal so as to achieve the required permanent oscillation. If conversely one wants to represent the controller as a continuous-time system, it is necessary to suitably coordinate it with the digital procedure.

The solution adopted here can be summarised as follows. First, implement the controller in the desired form (here, for consistence with the digital case, an antiwindup PI was chosen) as differential and algebraic equations. Then, realise the autotuning procedure as a digital algorithm, including the control computation during that procedure, exactly as it was in the fully digital case. Finally, manage the autotuning request event by (a) setting a flag that selects the control output to be that coming from the equations or the algorithm, depending on the mode, and (b) initialising the algorithm output to the last equation output. Analogously, manage the autotuning termination by resetting the above flag, and reinitialising the equation-based controller state to match the last algorithm output.

The only (small) disadvantage of such a solution is that the equation-based controller stays in place during the autotuning phase. However the resulting overhead is generally very limited, given the invariantly simple structure of the controller, while there is a gain in terms of simplicity with respect to possible alternative solutions attempting to avoid said overhead.

3.3 Hybrid version

The PI for this realisation is implemented in antiwindup form, i.e., as the block diagram of figure 4.

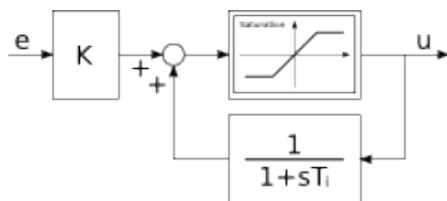


Figure 4: Block diagram of the used continuous-time antiwindup PI.

That scheme corresponds in Modelica to the equations

```
satIn = K*(SP-PV)+linFBout;
CSpi = Ti*der(linFBout)+linFBout;
CSpi = noEvent(max(CSmin,min(CSmax,satIn)));
```

where CSpi is the control signal in PI mode (u in figure 4), satIn the input of the saturation block, and

linFBout the output of the feedback block, added in the diagram to the term Ke .

Given all that, the Modelica model of the hybrid autotuning PI is shown in the listing below.

```
model ATPirelayNCmixedMode
import Modelica.Constants.*;
// ... same parameters as the fully digital version ...
Integer iMode;
Real K;
Real Ti;
Real satIn;
Real linFBout(start=0,stateSelect=StateSelect.always);
Real CSpi;
discrete Real CSat;
discrete Boolean AT;
discrete Boolean UP;
discrete Real rPVmax;
discrete Real rPVmin;
discrete Real rCSmax;
discrete Real rCSmin;
discrete Real lastToggleUp;
discrete Real period;
discrete Real wox;
discrete Real Pox;
discrete Integer nOx;
// ... same connectors as the fully digital version ...
equation
// Continuous-time antiwindup PI
satIn = K*(SP-PV)+linFBout;
CSpi = Ti*der(linFBout)+linFBout;
CSpi = noEvent(max(CSmin,min(CSmax,satIn)));
// Output selection
if iMode==0 or iMode==1 then // 0, PI or 1, AT init
    CS = CSpi;
else // 2, AT run
    CS = CSat;
end if;
algorithm
// Autotuning procedure
when initial() then
    K := K0;
    Ti := Ti0;
    AT := false;
end when;
when ATreq and sample(0,Ts) then // Turn on AT when required
    if not AT then
        AT := true; // set AT flag on
        iMode := 1; // set next mode to AT init
    end if;
end when;
when AT and iMode==1 and sample(0,Ts) then // AT init mode
    iMode := 2; // set mode to AT run
    CSat := pre(CSpi);
    UP := false;
    period := 0;
    wox := 0;
    Pox := 0;
    rPVmax := pre(PV);
    rPVmin := pre(PV);
    rCSmax := CSat;
    rCSmin := CSat;
    lastToggleUp := time;
    nOx := 0;
end when;
when (iMode==1 or iMode==2) and not AT
and sample(0,Ts) then // AT shutdown;
    iMode := 0;
    reinit(linFBout,CSat); // re-initialise the continuous-time PI
end when;
when AT and iMode==2 and sample(0,Ts) then // AT run mode
    if UP==false and PV<=SP then // Manage relay
        UP := true;
    end if;
    if UP==true and PV>SP then
        UP := false;
    end if;
    if UP==true then
        CSat := CSat + slope*Ts;
    else
        CSat := CSat - slope*Ts;
    end if;
    if PV>rPVmax then // record relay id max and min for PV and CS
        rPVmax := PV;
    end if;
    if PV<rPVmin then
        rPVmin := PV;
    end if;
    if CSat>rCSmax then
        rCSmax := CSat;
    end if;
    if CSat<rCSmin then
        rCSmin := CSat;
    end if;
    if UP==true and pre(UP)==false then // tune if perm ox
        period := time-lastToggleUp;
    end if;
end when;
end model
```

```

lastToggleUp := time;
if period>0 and n0x>=n0xMin
  and abs(period-pre(period))/period
  < perm0xPeriodPerc/100 then
  AT
  := false;
  wox
  := 2*pi/period;
  Pox
  := pi^2*(rPVmax-rPVmin)/8/(rCSmax-rCSmin);
  Ti
  := tan(pm/180*pi)/wox;
  K
  := tan(pm/180*pi)/(Pox*sqrt(1+(tan(pm/180*pi))^2));
end if;
rPVmax := PV;
rPVmin := PV;
rCSmax := CSat;
rCSmin := CSat;
n0x := n0x+1;
end if;
end when;
end ATPIrelayNCmixedMode;
    
```

Notice the presence of some `noEvent` clauses. In principle they can be omitted, but leaving them in slightly reduces the computational burden and, above all, is consistent with the operation of real-world autotuners, where inputs are typically acquired only at the beginning of a sampling period.

Also, observe how the proposed structuring can be quite easily generalised, including different tuning rules, different types of process stimulation (e.g., step-instead of relay-based) and different controller structures, since the presence of the autotuning *algorithm* does not modify in any sense the controller *equations*.

4 Simulation examples

In this section, two simulation examples are reported, to show the advantages yielded by the proposed autotuner models, and to verify their correct behaviour in realistic situations.

4.1 Example 1

This example aims at illustrating the correctness of the hybrid realisation, and its usefulness in terms of simulation efficiency.

The Modelica scheme used for the example is that of figure 5, where the ATPI block is the fully digital or the hybrid autotuning PI, alternatively.

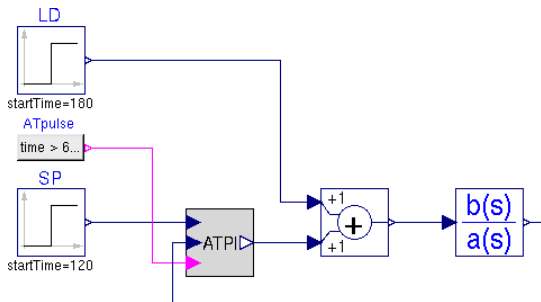


Figure 5: Modelica scheme for simulation example 1.

The process under control is described by the transfer function

$$P_1(s) = \frac{1}{(1+s)^3} \quad (5)$$

and the autotuning PI, in both the fully digital and the hybrid versions, is employed with a sampling time T_s of 0.1s, first leading the loop to steady state with a low-performance initial PI, then performing the autotuning operation with a required phase margin of 45° , and finally testing the so obtained PI with a set point and a load disturbance step, introduced respectively by the step sources SP and LD in figure 5.

Figure 6 shows the results, proving that the two realisations are *de facto* identical as for their outcome (in both cases, for example, the tuned PI has $K = 1.078$ and $T_i = 1.751$). On the other hand, however, the number of simulation steps required by the system with the hybrid autotuner in the 240s presented run is 3908, versus the 24007 of the system with the fully digital one. With so simple a process this does not turn into a significant reduction of the simulation time, but with more realistic a model of the controlled object, said advantage would be evident.

4.2 Example 2

This example shows the presented autotuner at work on a (slightly) more realistic example, namely the speed control of an axis, the model of which is built with standard Modelica blocks (with the sole exception of a noise generator) and is shown in figure 7. Three tuning operations are performed, with three different values of the required phase margin, namely 40° , 60° , and 80° .

Figure 8 shows the tuning results, obtained with the hybrid version of the autotuner (of course the fully digital one produces the same outcome). For brevity only the final part of the performed simulations is shown, when the PI is already tuned and the closed-loop system behaviour is tested by applying a set point step.

As can be seen, even in the presence of (reasonably) noisy measurements, the autotuning PI behaves correctly. It must be noticed that with the used tuning approach, the control system's cutoff frequency is dictated by the relay plus integrator experiment, as it clearly becomes ω_{ox} . For that reason, the relationship between the required phase margin and the *shape* of the obtained closed-loop transients, or even basic characteristics such as the maximum overshoot, is difficult to characterise in a formal way. Incidentally, this is especially true in the presence of resonances above the cutoff, which is typical of mechanical systems.

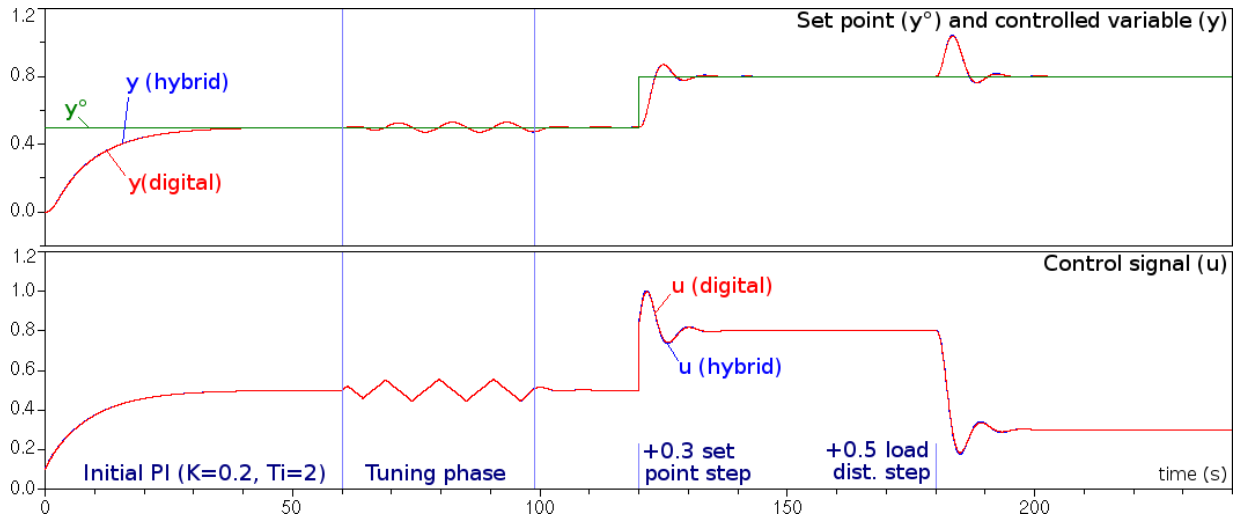


Figure 6: Closed-loop transients in simulation example 1.

Nonetheless, the prescribed phase margin is achieved, and in any case the mentioned difficulty is inherent to the employed autotuning approach, not to its Modelica representation. The interested reader can find in [8] a discussion on this matter.

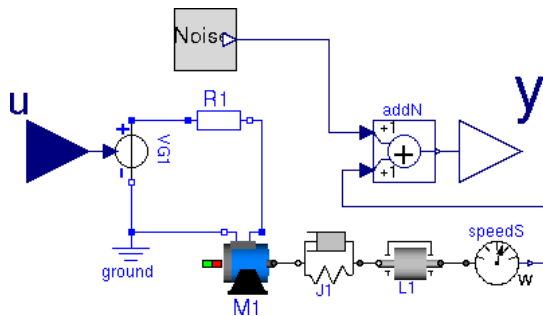


Figure 7: Model of the axis used in simulation example 2.

5 Some more words on the proposal usefulness

It was suggested above, as one of the motivations for this work, that a Modelica library of autotuners is useful to quickly set up the control system of a plant, or at least the part of it that is composed of PI(D) loops, and to verify the correct behaviour of a new autotuner by applying it in simulation to a benchmark set of models, conveniently chosen so as to represent the whole variety of applications where the new product is meant to be used.

After looking at the examples, and taking a more research-related point of view, at least one more use-

fulness can however be foreseen for such a library, and the underlying model structuring. Apologising in advance for the number of remarks to report prior to reaching the main point, the matter can be explained as follows.

In the first place, as can be noticed e.g. from the extensive review [17], establishing a taxonomy of tuning methods, also if the scope is restricted to a single controller structure, is far less trivial than one may expect.

Even more difficult is to set up a *comparative analysis* of such methods, basically because in the literature, when proposing and discussing a method, the process stimulation and information gathering phase is seldom accounted for. As shown in works like [14], comparisons between different tuning methods can be reversed by simply modifying the way in which the process is stimulated.

For the sake of completeness, it is worth observing that relay-based rules are the less keen to incur in that problem, since there is virtually no ambiguity on how process information is obtained and use with respect for example to the step-based identification of a fixed-order model, that can be carried out in a variety of manners, but nevertheless the problem exists, and needs addressing.

The absence of a taxonomy like that just envisaged is felt in the applications as an important open problem, see e.g. [11], because it makes it difficult to decide *a priori* which tuning rule is best suited for the particular problem at hand. In the opinion of the authors, the fact that a tuning method “sometimes works satisfactorily and sometimes does not”, with no apparent reason, is a major reasons for the resistances that autotuning still encounters in some applications. It is

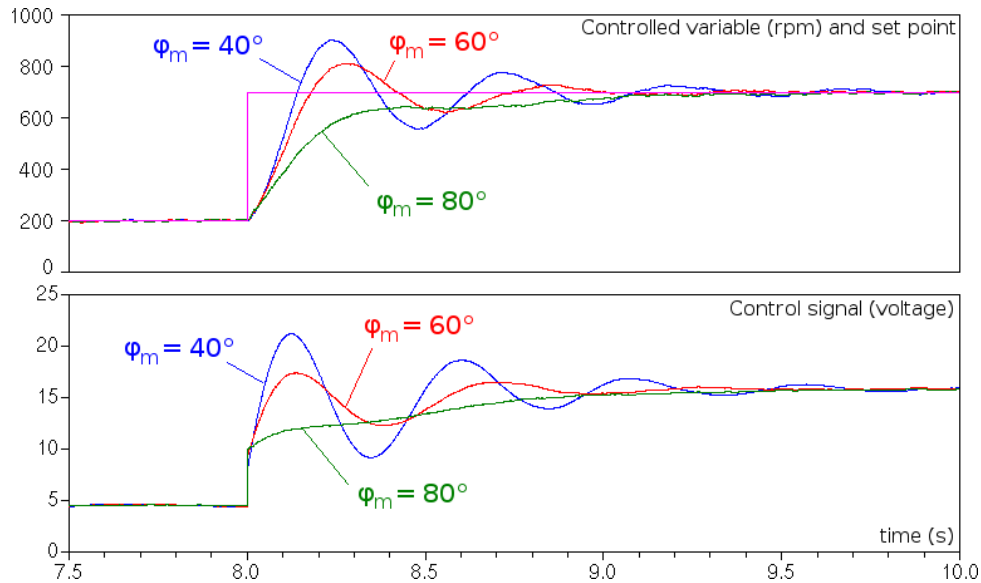


Figure 8: Tuning results in simulation example 2.

by definition possible to decide which rule (in a given and wide enough set) is the best for a given problem *a posteriori*, by simply applying all the rules in the set and examining the results, but this is clearly infeasible in practice.

As a result, most tuning rules are discussed “in nominal conditions”, i.e., making some structural assumptions on the process dynamics and performing the analysis under the hypothesis that the real process adheres to said assumptions [3].

Some attempts were made to circumvent the problem by means of the robust control theory, but this requires information on the *class* of processes to which the one under control belongs, and no matter how such a class has to be characterised, no *single* experiment can provide the necessary information. Attempts were also made to bring in the “identification for control” theory [6, 7], but unfortunately in many cases technological limitations do not permit to apply process inputs with the necessary excitation characteristics, and leave little (if any) room for “experiment design” as meant for in that theory.

For the problem just sketched, the presented library offers (part of) a solution. In fact, if evaluating a set of control rules *a posteriori* is infeasible in practice, it is not in simulation. Having in mind the type of application to be addressed (thermal, mechanical, and so forth), one can set up an enormous set of cases, test each considered rule on each case, and draw from such a simulation campaign the information required to set up a selection mechanism. In fact there are plenty of techniques to create such a mechanism, from interpo-

lation to soft computing [13, 10], and what is typically missing is precisely the data. On a similar front, when introducing a *new* tuning rule, the proposal is significantly strengthened if some idea is provided on how it will behave when coupled to realistic process experiments. Providing such information requires a lot of additional work with respect to the typical analyses performed in the literature, that are most frequently based on linear models, because in that domain is autotuning typically treated, and only the linear framework allows for powerful methods that do not require simulation.

As noticed e.g. in [19], however, the used models are frequently inadequate to examine the behaviour of an autotuner in the large, and therefore the mentioned analyses are sometimes confuted by experience, thereby further hindering a wide adoption of autotuning. Needless to explain why and how, the availability of a library like that presented here can help solve also this problem.

6 Conclusions and future work

The problem of simulating autotuning industrial controllers in Modelica was addressed, with the specific aim of obtaining efficient models. To this end, the controller is represented as a continuous-time system, while the autotuning procedure is realised as an algorithm. The proposed model structuring thereby allows to separate the two main parts of an autotuner clearly, preserving the simulation speed yielded by continuous-time control blocks, and replicating the autotuning software precisely.

As shown by the reported examples, and a number of other ones not reported here for space reasons, the so obtained simulation models are very precise if compared to fully digital ones, that certainly represent industrial implementation more closely, but oblige to pay for said fidelity in terms of simulation speed.

In this work, the focus was restricted to relay-based PI autotuning based on a single point of the process frequency response. It is however clear that the presented structuring is totally general, with respect to both the controller structure, the type of process stimulation, the tuning rules, and all in all the overall tuning procedure, inclusive of the logic needed to control the tuning operation. Future research will thus explore all those extensions, leading to a complete Modelica library of autotuning controllers, including different tuning rules and excitation procedures, and possibly addressing not only single controller blocks, but also the most frequently used control structures.

References

- [1] K.J. Åström and T. Hägglund. Automatic tuning of simple regulators with specifications on phase and amplitude margins. *Automatica*, 20(5):645–651, 1984.
- [2] K.J. Åström and T. Hägglund. Industrial adaptive controllers based on frequency response techniques. *Automatica*, 27(4):599–609, 1991.
- [3] K.J. Åström and T. Hägglund. *Advanced PID control*. Instrument Society of America, Research Triangle Park, NY, 2006.
- [4] A. Besançon-Voda and H. Roux-Buisson. Another version of the relay feedback experiment. *Journal of Process Control*, 7(4):303–308, 1997.
- [5] R.C. Dorf and H. Bishop. *Modern control systems*. Addison-Wesley, Reading, UK, 1995.
- [6] M. Gevers. Identification for control: from the early achievements to the revival of experiment design. *European Journal of Control*, 11(4–5):335–352, 2005.
- [7] H. Hjalmarsson. From experiment design to closed-loop control. *Automatica*, 43:393–438, 2005.
- [8] A. Leva. PID autotuning algorithm based on relay feedback. *IEE Proceedings-D*, 140(5):328–338, 1993.
- [9] A. Leva. Simple model-based PID autotuners with rapid relay identification. In *Proc. 16th IFAC World Congress*, Praha, Czech Republic, 2005.
- [10] A. Leva and F. Donida. Normalised expression and evaluation of pi tuning rules. In *Proc. 17th IFAC World Congress*, Seoul, Korea, 2008.
- [11] A. Leva and F. Donida. Quality indices for the autotuning of industrial regulators. *IET Control Theory & Applications*, 3(21):170–180, 2009.
- [12] A. Leva, S. Negro, and A.V. Papadopoulos. PI/PID autotuning with contextual model parametrisation. *Journal of Process Control*, 20(4):452–463, 2010.
- [13] A. Leva and L. Piroddi. Model-specific autotuning of classical regulator: a neural approach to structural identification. *Control Engineering and Practice*, 4(10):1381–1391, 1996.
- [14] A. Leva and L. Piroddi. On the parameterisation of simple process models for the autotuning of industrial regulators. In *26th American Control Conference (to appear)*, New York (USA), 2007.
- [15] Y. Li, K.H. Ang, and C.Y. Chong. Patents, software, and hardware for PID control—an overview and analysis of the current art. *IEEE Control Systems Magazine*, pages 42–54, february 2006.
- [16] W.L. Luyben. Getting more information from relay feedback tests. *Industrial & Engineering Chemistry Research*, 40(20):4391–4402, 2001.
- [17] A. O’Dwyer. *Handbook of PI and PID controller tuning rules*. World Scientific Publishing, Singapore, 2003.
- [18] R.C. Panda and C.C. Yu. Analytical expressions for relay feed back responses. *Journal of Process Control*, 13:489–501, 2003.
- [19] F.G. Shinskey. Process control: as taught versus as practiced. *Industrial & Engineering Chemistry Research*, 41(16):3745–3750, 2002.
- [20] T. Thyagarajan and C.C. Yu. Improved autotuning using the shape factor from relay feedback. *Ind. Eng. Chem. Res.*, 43:4425–4440, 2003.
- [21] C.C. Yu. *Autotuning of PID controllers: relay feedback approach*. Springer-Verlag, London, 1999.

Models of a post-combustion absorption unit for simulation, optimization and non-linear model predictive control schemes

J. Åkesson^{a,d}, R. Faber^b, C. D. Laird^c, K. Pröll^a, H. Tummescheit^a, S. Velut^a, Y. Zhu^c

^aModelon AB, Ideon Science Park, Lund, Sweden

^bVattenfall Research and Development AB, Berlin, Germany

^cArtie McFerrin Department of Chemical Engineering, Texas A&M University, U.S.A

^dDepartment of Automatic Control, Lund University, Sweden

Abstract

An increasing demand on load flexibility in power supply networks is the motivation to look at flexible, and possibly optimal control systems for power plants with carbon capture units. Minimizing the energy demand for carbon dioxide removal under these circumstances reduces the cost disadvantage of carbon capture compared to conventional production. In this work a dynamic model in Modelica of a chemical absorption process run with an aqueous monoethanolamine (MEA) is developed, and used for solving optimal control problems. Starting from a rather detailed dynamic model of the process, model reduction is performed based on physical insight. The reduced model computes distinctly faster, shows similar transient behavior and reflects trends for optimal steady-state operations reported in the literature. The detailed model has been developed in Dymola, and the reduced model is used in JModelica.org, a platform supporting non-linear dynamic optimization. First results are shown on the dynamic optimization of the desorption column, the main cause of energy usage in the process.

Keywords: CO₂, absorption, model, optimization, nonlinear model predictive control, Modelica, JModelica.org

1 Introduction

Carbon dioxide (CO₂) removal from a gas mixture using aqueous amine solutions is a well established process that previously has mainly been applied to gas sweetening of natural gas in refineries. Although the focus there lies primarily on the removal of hydrogen sulfide, it is equally applicable to flue gas from fossil-fuel fired power plants.

Figure 1 shows a schematic of the process. The CO₂ from the flue gas is absorbed by the liquid solvent in the absorber column. The cleaned gas is re-

leased to the environment, while the rich solution is pumped to the stripper column passing through a heat exchanger on the way. In the stripper at elevated temperatures, the CO₂ in the solution is released to a steam flow from the reboiler, which is driven by bled

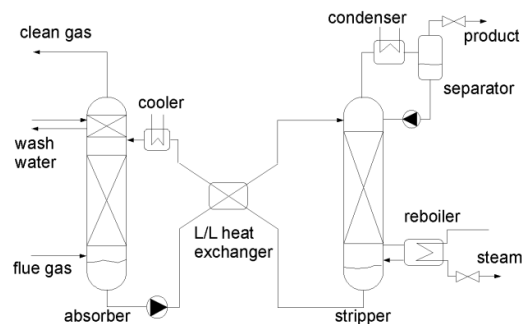


Figure 1: Schematic of an absorption/desorption process to remove carbon dioxide from power plant flue gas.

steam from the power generation process. Leaving the stripper at the top the product stream is after water separation compressed and stored. The overall power plant efficiency is expected to be reduced by at least 10 %, the solvent regeneration being responsible for more than half of this [1]. Minimizing the amount of steam required in the reboiler is therefore the task with highest priority in the optimization of this process.

With an increasing demand on the plant's flexible operation in the face of frequent load changes and an increased fraction of the generation capacity expected to come from renewables, dynamic simulation and optimization have become important tools to ensure an efficient incorporation of the carbon capture into the power generation. At the same time a trade-off must be found between efficiency losses and removal rate, possibly governed by time-varying economic boundary conditions.

This paper presents the preliminary results achieved within a larger project aiming at developing

an optimization technology for advanced model-based control of the separation plant. It focuses on the modeling of the capture plant, briefly presents the methods and tools that are used for optimization and presents preliminary results of solving an optimal control problem for the reduced model presented in the first half of the paper.

2 Background

2.1 Modeling of carbon dioxide removal with chemical absorption

System simulation models of amine scrubbing processes with different levels of detail can be found in the literature and as part of commercial toolboxes. The most rigorous models are developed for steady-state system computations with partial differential equations for mass transport along bulk flow and between the two phases, resulting in a high order system. This becomes easily too complex for dynamic system simulations, especially if parts of the power generation are supposed to be included or if used in model based control. Replacing rigorous models of multi-component mass transfer between gas and liquid with semi-empirical algebraic correlations reduces model complexity dramatically and is for example applied in [2] for an absorber description. Another model aspect with room for different levels of detail is the thermodynamic model of the liquid phase, describing the non-ideality of the electrolyte solution. Tobiesen compares in [3] a more rigorous with simpler approaches and concludes that high accuracy is rather a matter of a good data fit than model complexity.

Several studies on optimal operation of an amine-based CO₂ capture plant can be found in the literature. In [4] the effect of variables such as solvent circulation rate, stripper pressure or solvent temperature is investigated. The analysis is however static and considered only the variation of one parameter at a time, disregarding the multivariable and dynamic nature of the process. In [5] control strategies aiming at a fast response are developed using offline dynamic simulation of the process. In [6], both optimization and control of the plant are studied. The optimal conditions for operation are determined offline using static models and a suitable control structure to maintain the process close to optimal operation in spite of disturbances is thereafter derived using dynamic models.

The process industry has up to now not taken up the use of Modelica to the same degree as e.g. the automotive industry, mainly due a strong market

presence of domain-specific tools that are only applicable to process industry problems. Another important reason is the lack of physical properties for substances used in the process industry. There are, however, no other languages and tools that are as suitable as the combination of Dymola for high-performance simulation and JModelica.org for dynamic optimization for the given project, when the threshold of developing the fluid property models natively in Modelica is overcome.

2.2 Model Predictive Control

Model Predictive Control (MPC) is an advanced control method that relies of on-line solution of optimal control problems. During recent years, the method has become increasingly popular, especially in the process industry, [7]. The popularity of the method is attributed to its ability to handle multiple-input multiple-output (MIMO) systems, as well as control and state constraints. These two ingredients are common in a broad range of control problems. MPC allows the control engineer to tune a *cost function* to express the control objectives, typically by choosing weights in a quadratic cost function. By choosing the weights properly, the significance of the control objectives can be balanced. E.g., performance can be traded for robustness. In order to capture limitations in the plant to be controlled, *constraints* can be modeled. Constraints may represent tanks that may not over-flow or pressures that may not be exceeded for safety reasons. Other examples of constraints include limitations in actuators, such as limited ranges in valves and limited torques in motors.

In addition to a cost function and constraints, MPC relies on a *model* of the plant to be controlled. The model may be derived from first-principles, as is the case in this paper, or it may be computed from empirical data. Both linear and non-linear models can be used. During execution of the MPC controller, the model is used to predict the plant response to the future control inputs.

The key component of an MPC controller is the solution of an open loop optimal control problem (OCP). Based on the cost function, the constraints, the model and measurements, or estimates of the current plant state, optimal predicted trajectories for the model variables and the control inputs are computed. The first part of the optimal control variable trajectories is then applied to the plant. The procedure is then repeated periodically, each time shifting the optimal control horizon one step into further. This principle is called receding horizon control.

Solution of optimal control problems may be very computationally challenging, in particular for non-linear models. Application of MPC is therefore more

common in domains where typical plants have time constants in the range of minutes and hours rather than seconds. The CCS systems studied in this paper falls into this category, which makes MPC a feasible choice.

In addition to industrial use, MPC has also been extensively studied in the academic community, where a large body of theory has been developed, see, e.g., [11,12]. Notably, results for optimality, stability and robustness are available.

2.3 JModelica.org, Optimica and Dymola

In this work, Dymola is used as platform for simulation and as graphical editor while the software platform JModelica.org is used to solve dynamic optimization problems. The JModelica.org platform has been described earlier [8], and is currently undergoing rapid development both with respect to the parts of the Modelica language that are supported and with respect to the algorithms available. The main reason for choosing the JModelica.org platform is, however, that it offers strong support for solution of dynamic optimization problems, which is a key component of executing MPC controllers, as discussed above.

JModelica.org supports an extension of Modelica entitled Optimica [9], which allows dynamic optimization problems to be formulated based on Modelica models. Optimica enables the user to express cost functions, constraints, and what to optimize in a description format that is complimentary to Modelica's support for dynamic modeling using high-level language constructs. This feature enables shorter design cycles since more effort can be put into formulation of optimization problems rather than encoding them in a specialized format for a particular optimization algorithm. This property is valuable in this work, since extensive tuning of the cost functions and the constraints has proven necessary.

A direct collocation method, [10], is implemented in JModelica.org for solving large scale dynamic optimization algorithms. The method is applicable to differential algebraic systems and relies on full discretization of state, algebraic and control profiles. The resulting non-linear program is typically very large, but also sparse, which can be exploited by numerical software. In JModelica.org, the algorithm IPOPT, [11], is used to solve the NLPs resulting from collocation.

In terms of user interaction, JModelica.org offers a Python [12] interface. Using Python, Modelica and Optimica models can be compiled into executable optimization programs, optimization algorithms can be invoked and the results can be loaded. Python also comes with packages for numerical computations and visualization, which makes it a suitable environment for scientific computations. It can be noted

that the capabilities of Python go beyond scripting and atomization in that full-fledged applications with customized user interfaces can be created.

3 Dynamic model of an absorption/desorption column

The starting point in the development of a Modelica model suitable to be used in dynamic optimization is a model of an absorption unit developed in Dymola. The system consists of the main components absorber, stripper, reboiler and internal heat exchanger as well as auxiliary equipment such as pumps, valves, flow resistances, cooled vessels, sensors and reservoirs, as sketched in Figure 1. The solvent is an aqueous MEA solution.

Each packed section in a column consists of gas and liquid bulk flow and a static interface model describing the two-phase contact. Figure 2 shows the diagram layer of the packed section model. Gas and liquid phase are treated as separate media, each modeled as a separate medium property package. Thermodynamic equilibrium is only present at the phase interface, while mass and energy storage only occurs in the bulk flow.

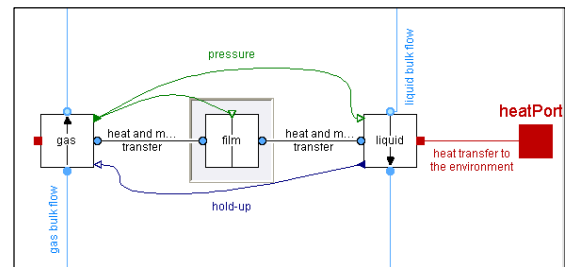


Figure 2: Diagram of the packed section model

Phase equilibrium at the gas-liquid interface for both, water and carbon dioxide, is computed as follows, assuming the pointing-factors and gas phase fugacity coefficients being equal to one.

$$y_{CO_2}p = \gamma_{CO_2}x_{CO_2}He_{CO_2} \quad (1)$$

$$y_{H_2O}p = \gamma_{H_2O}x_{H_2O}p_{H_2O,sat}(T) \quad (2)$$

with the mole fractions in gas and liquid phase y_i and x_i , the Henry-coefficient for dissolution of CO_2 in water He , the vapor pressure of water $p_{i,sat}$ and the system pressure p .

3.1 State selection

Pressure in the column is determined by the gas phase, with friction losses along the way through the packing material. The space available for the gas

phase is however dependent on the space occupied by the liquid phase. These properties and their derivatives are then passed to the respective other bulk component through signal connectors, see green and dark blue connections in Temperature and species amounts in each phase were chosen as independent state variables. Algebraic loops and high index problems can thus be avoided if

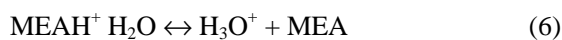
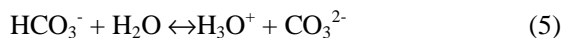
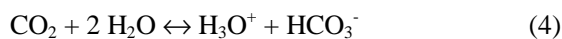
1. gas pressure can be directly computed from temperature and species amounts in the gas phase, e.g. using the ideal gas law or a cubic equation of state,
2. liquid density is independent of pressure (incompressible medium),
3. energy and species mass balances are formulated in terms of the derivatives of the chosen states,
4. and mass and heat transfer correlate concentrations and temperatures in the two dynamic volume models, gas and liquid bulk flow

Pressure drop in the gas phase and liquid hold-up are determined with literature correlations for packed columns, e.g. [Mackowiak], or user-defined nominal points, i.e. constant hold-up and gas flow operating point. The actual liquid hold-up correlates with the static set point via first order dynamics.

For a stripper column operated with MEA-solution and under the assumption that MEA is non-volatile, the number of dynamic degrees of freedom is then equal to 7 per volume segment (CO₂ gas, H₂O gas, CO₂ liquid, H₂O liquid, MEA, T liquid, T gas). In the absorber absorber additional flue gas component as oxygen and nitrogen are present. Column design, operation and demanded accuracy determine the required discretization of the packed sections in bulk flow direction, which usually is a number between 8 and 20.

3.2 Chemical reactions

The capacity of amines to absorb carbon dioxide is to a large extent based on chemical reactions. In the case of MEA as a solvent five main reactions can be identified as well as the zero charge condition.



This leads to a total of nine species in the liquid phase including the 6 ions. Throughout the models developed within this work, chemical equilibrium is assumed to be present, at the phase interface as well as in the bulk liquid. This assumption is thought to be justified at high temperatures as they are found in the stripper. The deviations resulting in the absorber are considered acceptable, if taking into account the poor availability of reliable kinetic data in the literature and the amount of additional dynamic states saved (5 per volume segment). However, a different solvent may demand a different approach.

3.3 Chemical equilibrium

The liquid phase speciation is determined by equilibrium constants K_j from the literature for each reaction j , which are determined empirically and expressed as polynomial functions of temperature. They are defined as

$$K_j = \prod (\gamma_i m_i)^{\nu_{i,j}} \quad (8)$$

where γ_i and m_i are the activity coefficient and molality of component i , respectively. $\nu_{i,j}$ is the stoichiometric coefficient of component i in reaction j , starting materials are considered with a negative sign, products with a positive one. Equilibrium constants allow also for an inference on heats of reaction, using the van't Hoff equation:

$$\frac{d \ln K}{dT} = \frac{\Delta H_r}{RT^2} \quad (9)$$

where ΔH_r is the enthalpy of reaction, T the temperature and R the ideal gas constant. The enthalpy of physical solution is computed accordingly using the temperature dependency of the Henry-coefficient [13].

However, a lot of computational time is required to solve the non-linear system of equations describing the speciation. Furthermore, extreme differences in ion concentrations by several orders of magnitude make a good choice of iteration variables essential for robust convergence.

In addition the following assumptions also apply:

- the flue gas entering the absorber contains only carbon dioxide, water, oxygen and nitrogen
- MEA is non-volatile and not present in the gas phase
- the total amount of liquid in the column is defined as the packing hold-up and the sump liquid volume

- the liquid in the column sumps and other large volumes is assumed to be ideally mixed
- mass and heat transfer between liquid and gas phase is restricted to the packed section
- negligible temperature difference between liquid bulk and interface to gas phase
- perfect gas law applies in the gas phase.
- phase equilibrium in reboiler and condenser

Table 1: References for physical properties used in the model

Property	Symbol	Used in reduced model	Reference
Equilibrium constants	K_i	indirectly	Collected in [14]
Henry-coefficient	He_{CO_2}	yes	[14]
Activity coefficients, liquid phase	γ_i	indirectly	[14]
Mass transfer coefficients	k_{iL}, k_{iV}	no	[15]
Diffusivities liquid phase	D_{iL}	no	[16] + Stokes – Einstein relation
Diffusivities gas phase	D_{iV}	no	Fuller's eq.in [17]
Densities and viscosities, liquid	ρ, μ	yes	[18]
Enhancement factor	E	no	[2]

The molecular carbon dioxide concentration $c_{CO_2,b}$ is then used to compute mass transfer between bulk and interface (if).

$$\dot{n}_{iL} = A_{if} k_{iL} E (c_{i,b} - c_{i,if}) \quad i = CO_2 \quad (10)$$

$$\dot{n}_{iV} = \frac{A_{if} k_{iV} (p_{i,b} - p_{i,if})}{RT} \quad i = CO_2, H_2O \quad (11)$$

where \dot{n}_{iL} and \dot{n}_{iV} denote the molar flows in the liquid and the vapor phase, respectively. A_{if} is the contact area, E is an enhancement factor describing the impact of chemical reactions on the concentration profile near the interface. k is a mass transfer coefficient, $c_{i,if}$ and $c_{i,b}$ are molar concentrations at the interface and in the liquid bulk, respectively and $p_{i,if}$ and $p_{i,b}$ are correspondingly partial pressures of the considered species in the gas phase. R and T are the ideal gas constant and bulk phase temperature, respectively.

Properties and correlations from the literature used in these models are listed in Table 1.

3.4 Model reduction

Online optimization as it is used in MPC implicates tighter limitations on the model size than pure dynamic simulation or even offline optimization would do. The solution of the optimization problem for a finite horizon needs to be found between two sampling instants and therefore demands a relatively low computational effort. But also the available memory to store result points for all model variables for each time step within the finite horizon limits the allowed number of algebraic and differentiated variables. However, exact numbers are hard to define in advance. At the same time accuracy demands are not as high as the model is updated with measurement values at each sample step.

The following measures are taken in order to reduce the model:

1. Chemical equilibrium computation (and ion speciation) was replaced by a spline approximation of the molecular CO_2 concentration in the liquid phase as a function of temperature and solvent loading with CO_2 . The mass fraction of MEA in the unloaded solution is kept constant at 30% for this function.

2. Enthalpy of absorption/desorption is replaced by a function of temperature but constant with solvent loading.
3. Mass transfer coefficients including enhancement by chemical reactions are no longer computed from physical medium properties, but become constant tuning parameters.
4. Reduction of the number of volumes in bulk flow direction to an acceptable minimum (iterative, dependent on application)
5. Constant specific heat capacities of all species and constant liquid density

3.5 Validation and model comparison

The total system model is composed of the two packed columns and complemented with washers, condensers, pumps and valves according to Figure 1. The reboiler, which supplies the gas flow to the stripper is modeled as a flash stage with phase equilibrium and uniform temperature. Simulation results of the detailed model are compared to experimental data from a pilot plant run with open control loops [19]. The input variables of the test case are:

- fluegas inlet flow and properties
- clean gas pressure
- liquid recirculation rate
- reboiler duty
- product stream outlet pressure

All inlet conditions are kept constant except for the flue gas rate, which is reduced by 30% after having run the plant in steady-state for some time. Figure 3

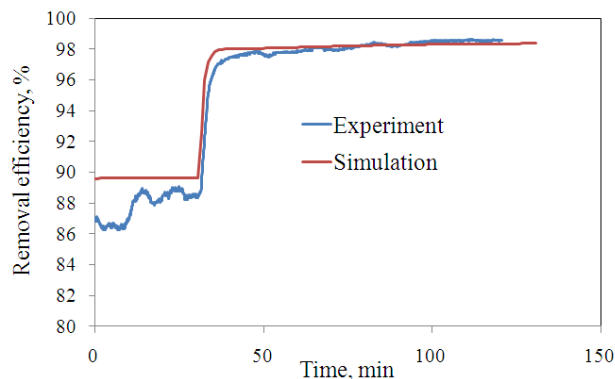


Figure 3: Carbon dioxide removal rate, experiment and simulation of the detailed model

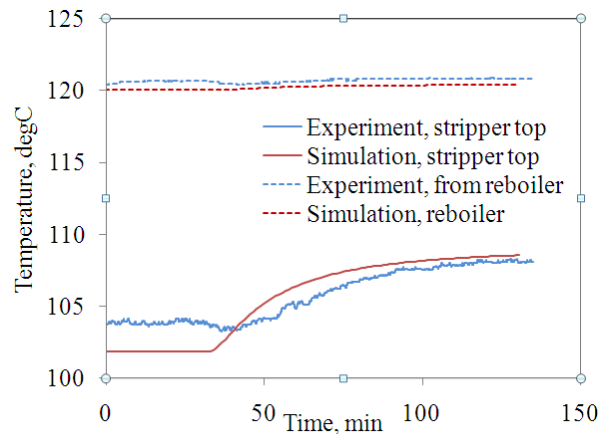


Figure 4: Stripper top and reboiler temperatures

shows the CO₂ removal rate before and after the step change in experiment and simulation. Giving the fact that the experiment apparently did not reach steady-state before the step, the agreement between the two curves is satisfactory.

The temperatures at the gas outlet of the stripper column and at the liquid outlet of the reboiler are plotted in Figure 4.

Especially the reboiler temperature, which is directly coupled to pressure and pressure drop along the gas flow path as well as the solvent loading, is in very good agreement with the experimental data.

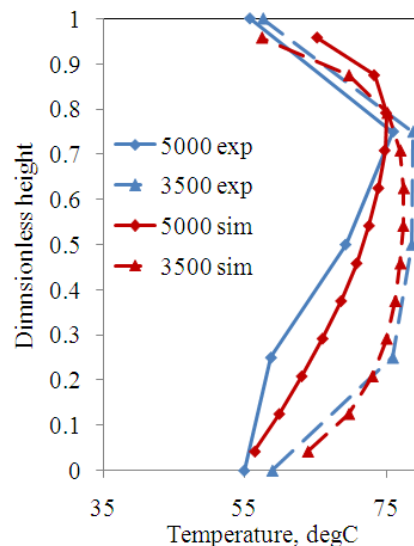


Figure 5: Temperature profile wrt column height

Since liquid phase concentration data is unavailable, it can be useful to look at temperatures instead, because of the direct connection between ab-

sorbed/desorbed carbon dioxide and temperature changes due to heats of reaction. Figure 5 compares the gas temperature profile along the absorber column height for experiment and simulation at presumed steady state before and after the flue gas step, respectively. The locations of the five measurement points were guessed to be equally distributed. The simulation captures well the location of the highest temperature first in the upper part and later with a lower gas flowrate as having moved further down.

The optimization problem in the next section is only solved for the stripper column including reboiler and condenser. A comparison of the detailed and the reduced model is therefore only performed for this part of the plant. Model assumptions, which affect the dynamic behavior of the unit, namely concerning liquid volumes and hold-ups, are similar in both models. Therefore, the comparison is restricted to steady-state operating points. Figure 6 presents the liquid lean loading at the stripper outlet as a result of reboiler duty under constant liquid inlet conditions and stripper top pressure. The results show that the energy required to regenerate the solvent to a certain lean loading is predicted close to each other with the two models. It can be concluded that the complexity of the reduced model is sufficient to investigate the energy consumption of the reboiler. The reduced model performed the stripper unit series 200 times faster than the detailed model. The simulations started at fixed initial states and simulated to steady-state. Large transients as they occur in the first seconds of a simulation demand especially large computational efforts, when using the detailed model.

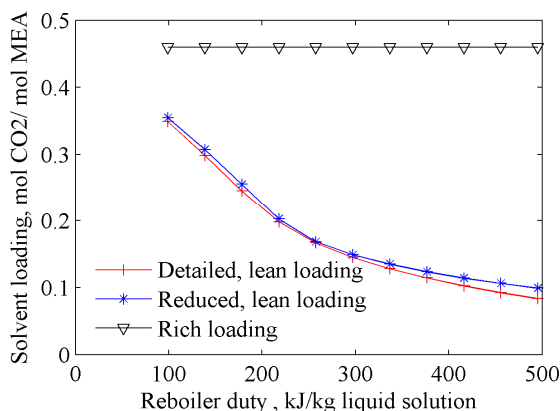


Figure 6: Solvent lean loading as a function of reboiler duty, detailed and reduced model

4 Optimization results

The goal of the project is to apply nonlinear model predictive control on the separation plant to minimize its energy usage. As it was mentioned in Section 2.2, this requires solving a sequence of open-loop optimal control problems. The aim of the present section is to show how those open-loop control problems can efficiently and accurately be solved using the simplified models and the tools previously described. For that purpose, a simple control problem using one of the most energy demanding parts of the separation plant, namely the stripper unit, will be formulated and solved.

4.1 Process model

The process to be optimized is the stripper unit shown in Figure 7.

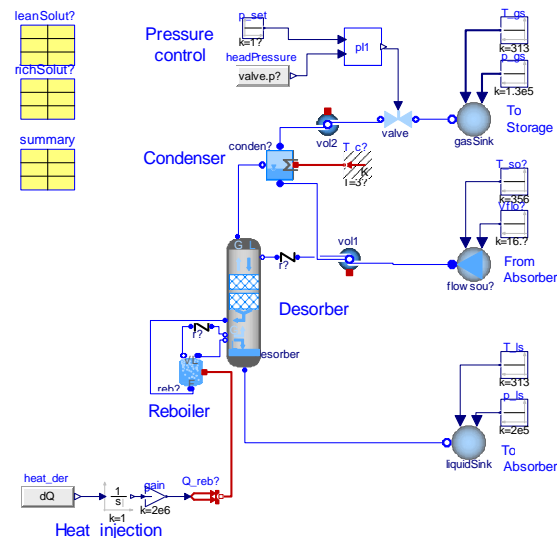


Figure 7: Graphical representation of the stripper unit used for optimization in Dymola

It is composed of:

- a reboiler
- a stripper column with packed sections and a sump
- a condenser to remove the water from the product stream
- a pressure control valve together with a pressure controller

The process model is described by 1493 equations and 1493 time-varying variables, including 50 continuous-time states. This is a larger model size than the size of the models reported in [20] for start-up optimization of coal fired power plants.

4.2 Control problem

Objective function. The control problem is formulated as in standard MPC using a quadratic cost function J penalizing deviations of the controlled variable y , as well as variations in the control signal u :

$$J(u, x_0) = \int_0^{H_p} \alpha (y(t) - y_{ref})^2 + \beta \left(\frac{du}{dt} \right)^2 dt$$

where α and β are weights that can be tuned to achieve a desired dynamic behavior and H_p is the prediction horizon.

Controlled variable. The variable to be controlled is the removal efficiency η of the separation plant. It is defined as the mass flow ratio of carbon dioxide leaving the condenser and carbon dioxide entering the absorber column with the fluegas:

$$\eta = \frac{\dot{m}_{\text{CO}_2, \text{condenser out}}}{\dot{m}_{\text{CO}_2, \text{absorber in}}}$$

Since the absorber column is not included in the optimization set-up, the CO_2 concentration in the rich solution entering the stripper column has been assumed to be in equilibrium with the flue gas entering the absorber column.

Control signal. The chosen control signal is the heat flow rate \dot{Q} to the reboiler. However, the decision variable of the optimization problem is chosen to be its time-derivative $\frac{d\dot{Q}}{dt}$, which is parameterized by a piecewise constant signal taking N values over the prediction horizon H_p , i.e. for $i=0..N-1$

$$\frac{d\dot{Q}}{dt}(t) = \gamma_i \in R, \quad t \in \left[i \frac{H_p}{N}, (i+1) \frac{H_p}{N} \right]$$

Only the first value of this open-loop optimization result, i.e. γ_0 would actually be applied to the process if the entire MPC algorithm was implemented.

Constraints. As far as the optimization constraints are concerned, they may be of both regulatory¹ and operational nature. The versatile JModelica.org plat-

form allows us to include any constraint that can be expressed in terms of process variables. In the present example, an upper limit on the reboiler pressure is imposed to avoid MEA degradation occurring at high temperatures.

$$p_{reboiler}(t) \leq p_{max}, \quad t \in [0, H_p]$$

A maximal temperature could equivalently be imposed since pressure and temperature are coupled in the reboiler.

Initial state. The initial state x_0 is assumed to be known and is computed using Dymola as the stationary point corresponding to a given heat flow rate $\dot{Q} = \dot{Q}_0$. An implementation of the MPC controller would require an observer to compute an estimate of the initial state x_0 based on the available measurements.

4.3 Numerical example

As mentioned in Section 2.3, the JModelica.org platform implements a direct collocation method to solve the optimal control problem. This implies that optimization is not performed on the continuous DAE system mentioned in 4.1, but on a discretized version using the Radau quadrature. The trajectory of every variable in the dynamic model is approximated by piecewise polynomials on each interval of the prediction horizon. In each interval, the approximation is exact at a number N_c of points, the collocation points. Choosing $N_c = 3$ and dividing the prediction horizon H_p in $N=10$ intervals of equal length converts the continuous optimization problem to an algebraic nonlinear program with 29824 variables, 29814 equality constraints and 5646 inequality constraints. Most of the inequality constraints originate from the max and min attributes associated to the physical variables. As the optimization problem is most probably non-convex, it is essential to provide the solver IPOPT with reasonable guessed trajectories for the initialization of the iterative optimization algorithm. The trajectories were here taken to be constant in time and given by the initial state x_0 , computed in Dymola.

A step change in the desired removal efficiency is now considered. Using the numerical values listed in Table 2, the optimization problem is solved in JModelica.org in 36 iterations. The results are shown in Figure 8.

¹ Regulatory rules for carbon capture plants are still under discussion, but will certainly play a role.

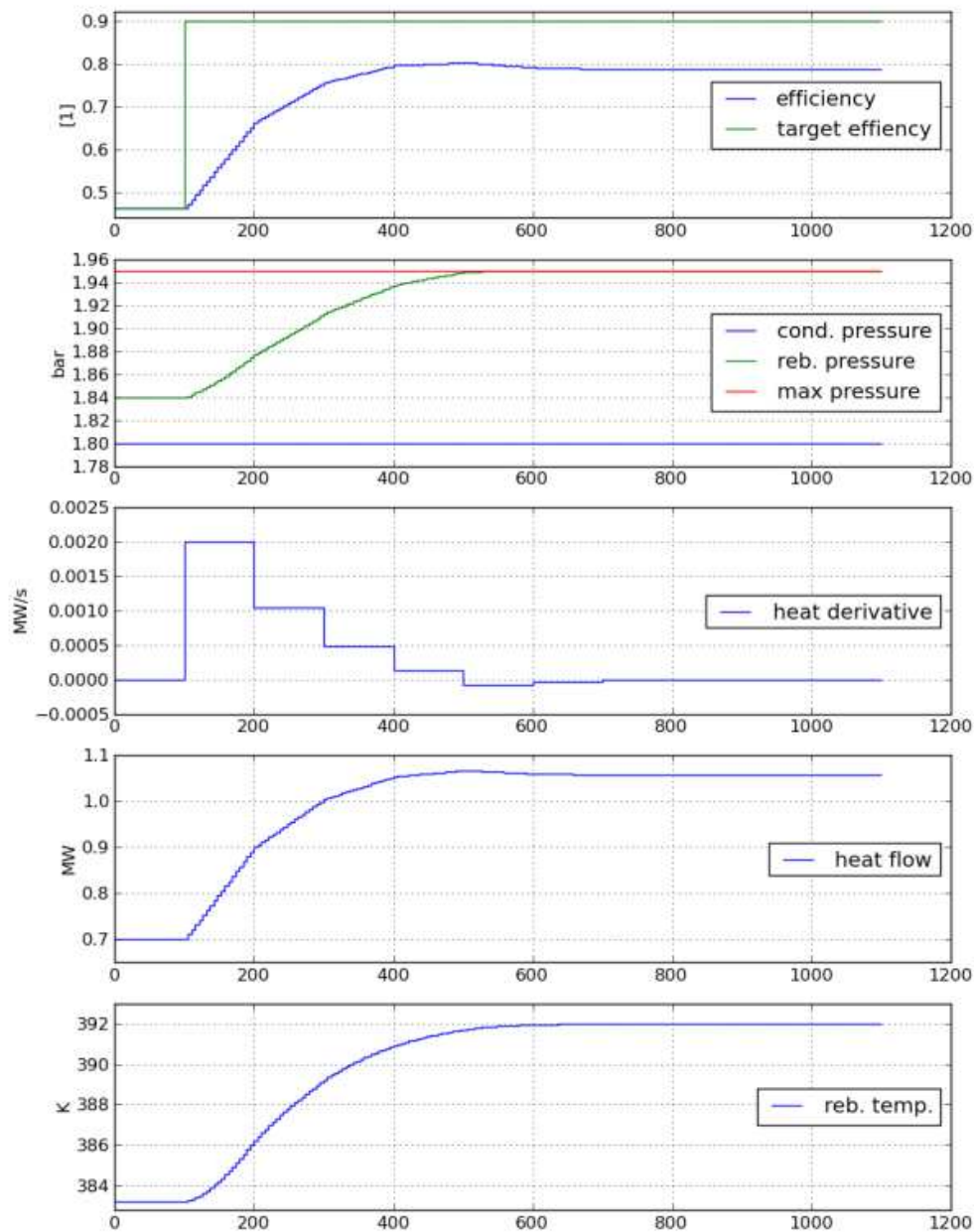
Table 2: Parameter values used in the optimization problem

1000s	0.1		0.7 M	1.95 bar	0.9
-------	-----	--	-------	----------	-----

At the beginning, the heat flow rate to the reboiler is rapidly increased from its start value of 0.7 MW to 1.05 MW, leading to a removal efficiency of about 0.8 at time $t=400$ s. At around 500s, the reboiler pressure reaches its maximal allowed value of 1.95 bar and the heat flow rate decreases slightly to avoid

constraint violation. Because of the high condenser pressure, the target efficiency of 0.9 cannot be achieved in this optimization setup. With a different column design or different boundary conditions, higher efficiency could of course be achieved.

To evaluate the consistency of the optimization result with respect to the continuous-time model equations, the optimized trajectories have been evaluated by applying the optimized heat input to the model implemented in Dymola. No difference could be observed when comparing results from JModelica.org and Dymola (results not shown).



er

5 Conclusions

A dynamic model of a post-combustion carbon capture process developed in Modelica was presented. The main focus lies on the chemical absorption of the carbon dioxide by the liquid solution in the absorber column. The same model can be used for the corresponding desorption process in the stripper column, by exchanging the flue gas medium for a mixture of water steam and carbon dioxide. A comparison of simulation results with experiments from a pilot plant showed a good agreement.

In a second step the model was reduced to meet the demands of a dynamic optimization. The largest performance improvement was achieved with a replacement of the chemical reactions in the liquid phase by an interpolated table with equilibrium data. A comparison of steady-state results from the stripper unit modeled with both approaches justified the usage of the reduced model for energy optimization purposes.

As a first step toward NMPC, a test case with the chosen system model was defined. It demonstrates the solution of an optimal control problem with the JModelica.org platform while adhering to specified variable constraints, in this case set on the reboiler pressure.

By formulating and solving this problem we have shown that the JModelica.org platform is a viable choice for solving large scale dynamic optimization problems, which is a prerequisite for NMPC applied to CCS plants. Future extensions include investigation of how to explore available control variables, cost function formulation, and state estimation.

6 Acknowledgements

This work was funded by the Swedish funding agency Vinnova under the grant program "Forska and Väx."

7 References

[1] Abu-Zahra MRM, Schneiders LHJ, Niederer JPM, Feron PHM, Versteeg GF. CO₂ capture from power plants. Part II. A parametric study of the economical performance based on mono-

ethanolamine. *International Journal of Greenhouse Gas Control* 2007;1:135-142.

[2] Kvamsdal HM, Jakobsen JP, Hoff KA. Dynamic modeling and simulation of a CO₂ absorber column for post-combustion CO₂ capture. *Chem Eng Process* 2009;48:135-144.

[3] Tobiesen FA, Juliussen O, Svendsen HF. Experimental validation of a rigorous stripper model for CO₂ post-combustion capture. *Chem Eng Sci* 2008;63:2641-2656.

[4] Freguia S, Rochelle GT. Modeling of CO₂ capture by aqueous monoethanolamine. *AIChE Journal* 2003;49(7):1676-1686.

[5] Ziaii S, Rochelle GT, Edgar TF. Dynamic modeling to minimize energy use for CO₂ capture in power plants by aqueous monoethanolamine. *Ind Eng Chem Res* 2009;48:6105-6111.

[6] Panahi M, Karimi M, Skogestad S, Hillestad M, Svendsen HF. Self-optimizing and control structure design for a CO₂ capturing plant. In: Eljack FT, Rex Reklaitis GV, El-Hawagi MM, editors. *Proceedings of the 2nd Annual Gas Processing Symposium, Qatar 2010*.

[7] Qin SJ, Badgwell TA. A survey of industrial model predictive control technology. *Control Engineering Practice* 2003;11: 733-764.

[8] Åkesson, J., Årzén, K.-E., Gäfvert, M., Bergdahl, T., & Tummeseit, H. Modeling and Optimization with Optimica and JModelica.org—Languages and Tools for Solving Large-Scale Dynamic Optimization Problem. *Computers and Chemical Engineering* 2010, Doi:10.1016/j.compchemeng.2009.11.011.

[9] Åkesson, J. Optimica—an extension of Modelica supporting dynamic optimization. 6th International Modelica Conference 2008.

[1] Biegler LT, Cervantes A, Wächter A. Advances in simultaneous strategies for dynamic optimization. *Chemical Engineering Science* 2002;57:575-593.

[11] Wächter A, Biegler LT. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 2006;106 (1); 25-58.

[12] Python Software Foundation. Python Home Page. Retrieved from www.python.org, 2010.

[13] Kim I, Hesssen ET, Haug-Warberg T, Svendsen HF. Enthalpies of Absorption of CO₂ in Aqueous Alkanolamine Solutions from e-NRTL Model. *Energy Procedia* 2009;1:829-835.

[14] Böttinger W. NMR-spektroskopische Untersuchung der Reaktivabsorption von

- Kohlendioxid in wässrigen Aminlösungen. Dissertation. Universität Stuttgart; 2005
- [15] Onda K, Takeuchi H, Okumoto Y. Mass transfer coefficients between gas and liquid packed columns *J Chem Eng Jpn* 1968;1(1):56-62
- [16] Versteeg GF, van Dijk LAJ, van Swaaij WPM. On the kinetics between CO₂ and alkanolamines both in aqueous and non-aqueous solutions. An overview. *Chem Eng Commun* 1996;144:113-158.
- [17] Poling BE, Prausnitz JM, O'Connell JP. The properties of gases and liquids. 5th edition. New York: McGraw-Hill;2001
- [18] Weiland RH, Dingmann JC, Cronin DB, Browning GJ. Density and Viscosity of Some Partially Carbonated Aqueous Alkanolamine Solutions and Their Blends. *J. Chem. Eng. Data* 1998;43(3):378–382
- [19] Faber R, Köpcke M, Biede O, Nygaard-Knudsen J, Andersen J. Open-loop step responses for the MEA post-combustion capture process: Experimental results from the Esbjerg pilot plant. Proceedings of the GHGT-10 conference 2010.
- [20] Franke, R. Doppelhamer, J., Online Application of Modelica Models in the Industrial IT Extended Automation System 800xA, Proceedings of 5th International Modelica Conference, 2006, Vienna, Austria.

Robust Initialization of Differential-Algebraic Equations Using Homotopy

Michael Sielemann¹, Francesco Casella², Martin Otter¹, Christoph Clauß³, Jonas Eborn⁴,
Sven Erik Mattsson⁵, Hans Olsson⁵

¹DLR Institute of Robotics and Mechatronics, Germany

²Dipartimento di Elettronica e Informazione, Politecnico di Milano, Italy

³Fraunhofer EAS, Dresden, Germany

⁴Modelon AB, Lund, Sweden

⁵Dassault Systèmes AB, Lund Sweden

Abstract

The new operator homotopy(.) was introduced in Modelica 3.2 to improve the solution of difficult initialization problems. The background and motivation for this approach is discussed and it is demonstrated how to apply it for mechanical, electrical and fluid systems. Furthermore, it is shown at hand of several examples how an inappropriate formulation might lead to ill-posed problems.

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 (*Mattsson et al., 2002*). 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:

$$\mathbf{0} = \mathbf{f}(\dot{\mathbf{x}}, \mathbf{x}, \mathbf{w}, t), \quad \mathbf{x}(t) \in \mathbb{R}^{nx}, \quad \mathbf{w}(t) \in \mathbb{R}^{nw}, \quad t \in \mathbb{R}.$$

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, 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 $2*nx+nw$ unknowns and the DAE has $nx+nw$ unknowns, additional nx equations must be provided which are called “initial equations” in Modelica:

$$\mathbf{0} = \mathbf{g}(\dot{\mathbf{x}}_0, \mathbf{x}_0, \mathbf{w}_0, t_0), \quad \dim(\mathbf{g})=nx$$

The most often used initial equations are:

$$\dot{\mathbf{x}}_0 = \mathbf{0}$$

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, 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. Library developers therefore typically implement approximate equations

in order to set these. They are usually formulated in terms of parameters (of e.g. the boundary conditions). At the same time, the iteration variables of the nonlinear equation systems may change after small topological modifications to the model. 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.

The goal of this contribution is to provide the solver with more information on the problem to solve. This is performed in an object-oriented way and seamlessly integrates with the concept of equation-based object-oriented languages.

2 Nonlinear Equation Solvers and Homotopy

The classic gradient-based iterative algorithms to solve nonlinear algebraic equation systems such as damped Newton's Method provide local convergence only, see, e.g., (Dennis and Schnabel, 1996), (Deufhard, 2004), (Kelley, 2003). Such algorithms may fail due to various reasons such as the residuals not being Lipschitz continuously differentiable or containing local minima with respect to some norm introduced by the algorithm.

Several alternatives to these conventional methods exist. Homotopy is one of them and is considered in this contribution to meet the need for more robust initialization.

2.1 Established Homotopy Methods

In homotopy methods for solving nonlinear algebraic equation systems, the idea is to start with a simplified problem and continuously deform it to the difficult problem of interest. Even though this appears to be conceptually simple, several details of these methods and algorithms have to be considered. Unless certain prerequisites are met, the existence of the homotopy trace between the start and a solution, finite length of the path, nonexistence of singularities along the path and other important requirements are not guaranteed.

The homotopy is constructed from a system of residual equations that is easy to solve, as well as the one of interest, $\mathbf{F}(\mathbf{z}) = \mathbf{0}$. Here, a generic vector \mathbf{z} of unknowns is used. In the Modelica case, this vec-

tor is: $\mathbf{z} = [\dot{\mathbf{x}}_0; \mathbf{x}_0; \mathbf{w}_0]$ and the equations are $\mathbf{F} = [\mathbf{f}; \mathbf{g}]$. The homotopy is then a system of equations with one higher dimension and is denoted by

$$\boldsymbol{\rho}(\mathbf{z}, \lambda) = \mathbf{0}.$$

The additional dimension is the homotopy or continuation parameter λ . It is typically restricted to the range $0 \leq \lambda \leq 1$ such that $\boldsymbol{\rho}(\mathbf{z}, 0) = \mathbf{0}$ is solved easily and $\boldsymbol{\rho}(\mathbf{z}, 1) = \mathbf{F}(\mathbf{z})$ is the system of interest.

At least three different homotopies are discussed in literature. We introduce the *Fixed Point Homotopy* following (Chow et al., 1978) as

$$\boldsymbol{\rho}(\mathbf{z}, \lambda) = \lambda \cdot \mathbf{F}(\mathbf{z}) + (1 - \lambda) \cdot (\mathbf{z} - \mathbf{z}_0).$$

Here, \mathbf{z}_0 is the start iterate. According to (Keller, 1978), the *Newton Homotopy* (or Global Homotopy) is defined as follows:

$$\begin{aligned} \boldsymbol{\rho}(\mathbf{z}, \lambda) &= \lambda \cdot \mathbf{F}(\mathbf{z}) + (1 - \lambda) \cdot (\mathbf{F}(\mathbf{z}) - \mathbf{F}(\mathbf{z}_0)) \\ &= \mathbf{F}(\mathbf{z}) - (1 - \lambda) \cdot \mathbf{F}(\mathbf{z}_0) \end{aligned}$$

Finally, the *Affine Homotopy* is introduced following (Wayburn and Seader, 1987) as

$$\boldsymbol{\rho}(\mathbf{z}, \lambda) = \lambda \cdot \mathbf{F}(\mathbf{z}) + (1 - \lambda) \cdot \mathbf{F}'(\mathbf{z}_0) \cdot (\mathbf{z} - \mathbf{z}_0)$$

Here, $\mathbf{F}'(\mathbf{z}_0)$ denotes the Jacobian of the residual equations at the start iterate.

The Newton Homotopy has the advantage of scale-invariance (Wayburn and Seader, 1987). However, the simple problem $\boldsymbol{\rho}(\mathbf{z}, 0) = \mathbf{0}$ may have several solutions and infinite loops that do not cross $\lambda = 1$ may result. Such tracks are called isolae (Choi and Book, 1991). The Fixed Point and Affine Homotopies only contain a single solution to the simple problem. Therefore, starting continuation inside an isola is impossible. The Affine Homotopy is also scale-invariant and the Fixed Point Homotopy is not (Wayburn and Seader, 1987).

Affine and Fixed Point Homotopies in turn may prescribe traces, which diverge toward an infinite value of some elements of the unknowns \mathbf{z} . Obviously, such traces cannot be followed numerically as the arc length is infinite and because the sign may change.

We note that (Chow et al., 1978) provides theorems on the success of the Fixed Point Homotopy with probability one in the sense of a Lebesgue measure. Success means that the track is of finite length, bounded and free of singularities (with the exception of turning points, which are not critical). The associated coercivity conditions on the residual equations were successfully employed in the area of analog circuit simulators for example. However, it is

neither possible to translate the boundedness condition on the solution of the residual equations $\mathbf{F}(\mathbf{z})$ nor the Inner Product Condition on $\mathbf{F}(\mathbf{z})$ to general multi-domain physical modeling as needed by a Modelica model. The former is the case due to the existence of unsaturated amplifier components, which arise in several applications, and the latter due to the lack of energy dissipation in component models to compensate the effect of boundary conditions in several physical domains other than electronics.

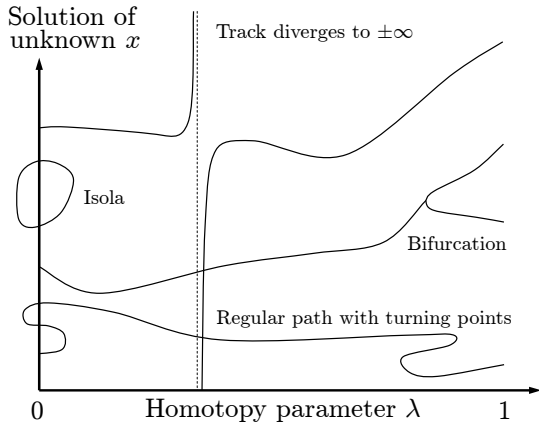


Figure 1: Illustration of failure modes of homotopy

In summary, the experience of the authors shows that such established homotopy methods are not sufficiently robust due to the above mentioned failure modes. Furthermore, an implementation of the Fixed Point Homotopy within Dymola 7, which was available since several years, did not provide indications of increased robustness of this approach with respect to the Newton solver in practical applications.

2.2 General Problem-Specific Homotopy

The issues in the general homotopies introduced so far stem from continuously deforming two rather unrelated systems of equations into each other. In the case of the Fixed Point Homotopy the simplified problem is the linear system $\mathbf{z} - \mathbf{z}_0$.

The source of the problem is the “lack of additional information” that can be utilized for the solution. In order to improve this situation for Modelica, a problem-specific homotopy is introduced:

- By deriving the simplified system from the actual system of interest, and
- by formulating the simplified system such that a homotopy to the actual problem of interest be free of singularities.

The formulation of the simplified system is problem-specific and allows modelers to infuse their knowledge about the physics of the problem into the way the equation system is solved (cf. Introduction). The approach is compatible with object-orientation and declarative modeling and is understood as something

introduced by domain experts to selected key equations. The goal is the formulation:

$$\boldsymbol{\rho}(\mathbf{z}, \lambda) = \lambda \cdot \mathbf{F}(\mathbf{z}) + (1 - \lambda) \cdot \tilde{\mathbf{F}}(\mathbf{z}).$$

Here, $\mathbf{F}(\mathbf{z})$ is the actual problem and $\tilde{\mathbf{F}}(\mathbf{z})$ is the simplified one. Based on a proposal by M. Otter, M. Sielemann and F. Casella, the new built-in operator, `homotopy(...)` was introduced in Modelica 3.2. It depends on two arguments, namely `actual`, the Real expression describing the actual problem, and `simplified`, the Real expression corresponding to the simplified problem. The Modelica translator can then expand this operator according to the homotopy. For the homotopy given above, which will be used throughout the remaining part of this article, the expression

$$\text{homotopy}(\text{expr1}, \text{expr2})$$

is thus expanded to

$$\lambda \cdot \text{expr1} + (1 - \lambda) \cdot \text{expr2}.$$

In contrast to other language constructs, the benefit of using this operator is that only one equation system for any number of steps is needed for initialization, and that it is logically defined how to transform one equation system into the other.

3 Implementation in Modelica Tools

The implementation of the new homotopy operator in a Modelica tool is rather straightforward: During the symbolic manipulation phase (BLT transformation, Pantelides algorithm etc.), the operator is treated as a function with two arguments. When generating code, the tool has to conceptually perform one homotopy iteration over the whole model and not several homotopy iterations over the respective local algebraic equation systems. The reason is that the following structure can be present:

$$\mathbf{w}_1 = \mathbf{f}_1(\mathbf{x}) \quad // \text{ has homotopy operator}$$

$$\mathbf{0} = \mathbf{f}_2(\dot{\mathbf{x}}, \mathbf{x}, \mathbf{w}_1, \mathbf{w}_2)$$

Here, a local non-linear equation system \mathbf{f}_2 is present. The homotopy operator is, however, used on a variable that is an “input” to the non-linear algebraic equation system and modifies the characteristics of it. The only useful way is to perform the homotopy iteration over \mathbf{f}_1 and \mathbf{f}_2 together.

This approach is “conceptual”, because more efficient implementations are possible, e.g. by determining the smallest iteration loop, that contains the equations of the first BLT block in which a homotopy operator is present and all equations up to the last BLT block that describes an equation system.

Various continuation algorithms have been suggested in literature, which are all suitable to trace homotopies of the type considered herein (e.g. pseudo arc-length algorithms). Popular examples are Hompack (Hompack, 2010) and Alcon2 (Elib, 2010; Deuffhard et al. 1987).

In order to validate the methodology, a test implementation was developed by M. Sielemann, which utilizes the Dymola software (Dymola 2010) and the Loca continuation algorithms of Trilinos (Heroux et al., 2005). One practical advantage of Loca over Hompack and Alcon2 is that the sensitivities of the homotopy with respect to the continuation parameter λ do not have to be provided. For Hompack and Alcon2 this has to be provided and had to be implemented using finite differences. The Loca/Dymola implementation has the following features:

- It provides three options for the treatment of the suggested homotopy operator. Normally, it is expanded to the given homotopy expression. Alternatively, simplified equation sets are obtained by inlining either argument. In case of the simplified argument, maximum structural simplifications of the equation system result.
- The user is able to manually prescribe whether to use homotopy initialization or not. This is an important feature for library development and debugging, and may be useful for users, too (e.g. if a local gradient based solver converges to a mathematically valid, but physically unreasonable solution or when a local gradient based solver does not converge and a user does not want to wait at the start of each simulation until the software realizes this and switches to homotopy initialization).
- Verbose information on the homotopy is optionally provided, which is useful for library development and debugging. In particular, the homotopy traces are visualized. Like this, it is possible to reconstruct what happens during the solution of the simplified problems and the homotopy transformation.

Additionally, Dymola 7.5 Beta also supports the homotopy operator. It was used for some of the application examples.

4 Application Examples

In this section several examples are given how to utilize the homotopy operator in different physical domains in order to solve difficult initialization problems.

4.1 Mechanical Systems with Kinematic Loops

Whenever kinematic loops are part of a mechanical system, non-linear algebraic equation systems are present. If these equation systems are solved numerically, the user has to provide guess values for the iteration variables in order that the system can be initialized. The issues are first demonstrated at hand of a simple example, the four bar mechanism, see Figure 2:

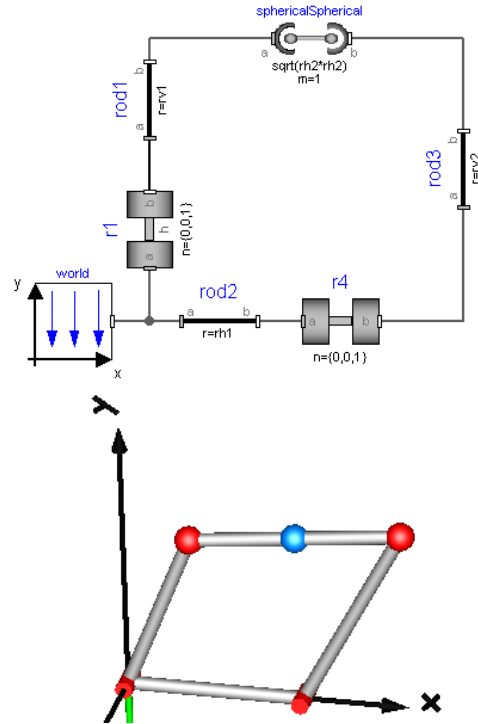


Figure 2: Four bar mechanism (top: Modelica model, bottom: animation).

The four bar mechanism consists of 4 connected revolute joints where the rotation axes of the joints are all parallel to each other. Since this mechanism is over constrained (e.g., the forces perpendicular to the kinematic loop cannot be uniquely determined), the upper two revolute joints are replaced by spherical joints which gives the same kinematic motion, but all quantities can be uniquely computed. Since joint r1 shall be driven by a drive train, the angle of this joint, “r1.phi” and its derivative are defined to be states by selecting in the “Advanced” menu of joint “r1” the option “stateSelect = StateSelect.always”.

This mechanical system gives rise to 9 nonlinear algebraic equations that are transformed by Dymola to one non-linear algebraic equation in one unknown. This equation is the constraint that the distance between the two spherical joints is constant. Formally, this nonlinear algebraic equation has the form:

$$0 = f(r1.phi, r4.phi)$$

where r1.phi is the “known” state and “r4.phi” is the angle of the right lower revolute joint that is used as

iteration variable. This nonlinear equation has two solutions that correspond to the two configurations of the mechanism. In order to initialize this mechanism, a “guess” value for variable $r4.\phi$ has to be provided.

It is always a useful strategy to define a mechanism in a reference configuration in which all generalized joint coordinates are zero and where all relevant kinematic quantities can be easily determined. When the mechanism is initialized in this way, the nonlinear equations of the initialization problem are fulfilled. In the case of the four bar mechanism, the selected reference configuration (in which $r1.\phi = r4.\phi = 0$) is selected such that the left bar is directed along the y-axis and the lower bar along the x-axis, respectively (see left part of Figure 3 below).

Problems arise, if the simulation of the mechanism shall not start in the reference configuration, but at a user-defined angle $r1.\phi = \phi_0$. Depending on the “guess” value of “ $r4.\phi$ ” the numerical solver might no longer find a solution, or if it computes a solution, it might be the wrong configuration.

In the example of Figure 3, a guess value of $r4.\phi = 45^\circ$ is selected and $r1.\phi$ is changed from $r1.\phi = 0^\circ$, in steps to -20° . The initial solutions found by Dymola are shown in Figure 3:

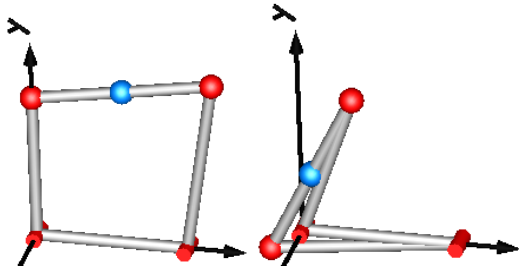


Figure 3: Initial solutions: (left: $r1.\phi = 0^\circ$, right: $r1.\phi = -20^\circ$)

Starting at about 18.9° the configuration is changing to an undesired configuration. This type of initialization is not robust, since for every change of the initial states, all guess values need to be properly adapted, which is usually difficult (not practical) if the system is no longer in its reference configuration.

The homotopy operator opens up a completely new direction: In the model of the revolute joint, the equation for the joint angle is changed to

```

if homotopyInitialization then
  angle = phi_offset + homotopy(phi,0);
else
  angle = phi_offset + phi;
end if;

```

where `homotopyInitialization` is a Boolean parameter that is set to **true** for $r1$ and set to **false** for $r4$. Furthermore, the start value of $r4.\phi = 0$ (the value from the reference configuration). The meaning is that independently which start value is given

for $r1.\phi$, the mechanism is initialized in its reference configuration $r1.\phi = 0$ (where the nonlinear algebraic equation is identically fulfilled) and then $r1.\phi$ is moved by the homotopy method until it reaches its start value. In every iteration a good guess value exists from the previous step and therefore the nonlinear equation is solved and remains in the configuration of the reference configuration. As a result, a very robust initialization of the mechanism is obtained, see Figure 4:

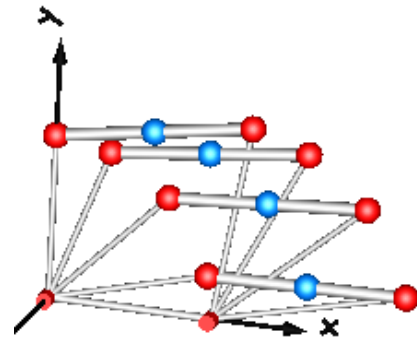


Figure 4: Initial solutions for $r1.\phi.start = 0^\circ, -20^\circ, -45^\circ, -75^\circ$

The four bar mechanism was only introduced to demonstrate the issues on a simple mechanism¹.

The sketched initialization technique shall now be applied on a much more involved example: A “Delta” robot (Clavel 1990). This robot is commercially available by several companies, e.g., by ABB under the name “FlexPicker™”². A suitable reference configuration of this robot is shown in Figure 5:

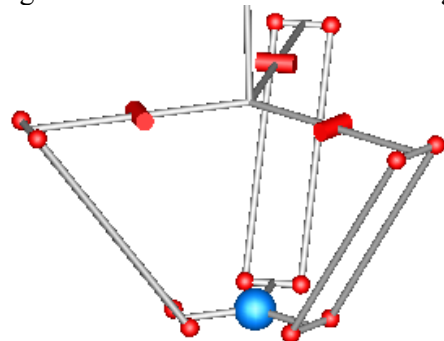


Figure 5: Delta robot in its reference configuration.

At the top, the robot consists of 3 actuated revolute joints that each drives a parallelogram. Every parallelogram consists of 4 spherical joints. In the bottom, the three parallelograms are rigidly mounted on a plate (in the figure visualized by a blue sphere that marks the center of mass of the load body that is attached to this point; in commercial robots, there is

¹ The equation system can be solved analytically when using an `Assemblies.JointRRR` joint from the `Modelica.Mechanics.MultiBody` library.

² FlexPicker is a trademark of ABB.

usually an additional revolute joint here). Overall, this robot has 3 revolute joints, 12 spherical joints and has 6 coupled kinematic loops. The robot has 3 degrees of freedom and can be controlled by the 3 revolute joints. By construction, the load plate is always parallel to the mounting plate on top, independently of the actual joint angles. Within its workspace, the robot can move very fast to a desired position. Since the motors that are mounted on the top plate are not moved, accelerations can be up to 30 g and speeds of 10 m/s can be reached.

Both direct kinematics (= given the joint angles, compute the position of the load), as well as the inverse kinematics (= given the load position, compute the joint angles) give rise to nonlinear algebraic equation systems. The more complicated case is the direct kinematic solution. When the robot is built up with “Joints.SphericalSpherical” joints (that each introduces a length constraint between two spherical joints), then Dymola transforms the system of 87 nonlinear algebraic equations down to 6 equations. If the joint angles are given, the resulting equation system has 16 configurations, but only the one shown in Figure 5 is the desired one. With the homotopy initialization, this system is initialized in the following way:

1. In the reference configuration, the absolute position $r[3]$ of the center point of the load plate, as well as the rotation angles $\phi[3]$ from the inertial frame to the load frame can be easily analytically computed ($r = \{0, 0, -\sqrt{L^2 - (r_1 + r_2 - r_3)^2}\}$, $\phi = \{0, 0, 0\}$). These values are provided as start values to the load body (since Dymola selects them as iteration variables of the nonlinear equation system).
2. The homotopy initialization of the revolute joints is switched on. So, for given start angles, the robot always starts first in the reference configuration and then moves the angles to the desired start configuration.

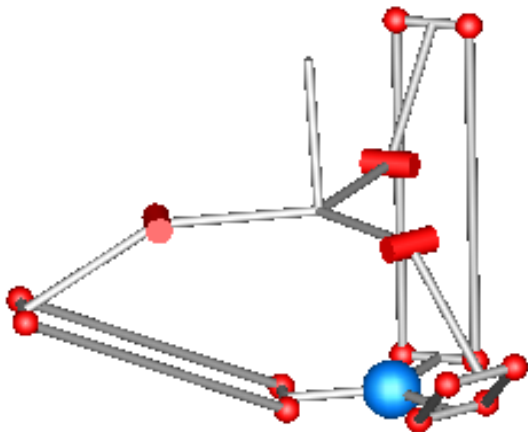


Figure 6: Delta robot initialized in configuration $\{45^\circ, -45^\circ, 30^\circ\}$.

Practical experience shows that within the technical workspace of this robot, the initialization is very robust. A typical example is shown in Figure 6.

The path of the three position variables of the load mass as function of the homotopy parameter (computed with Loca) is shown in Figure 7. As can be seen, the three paths are nearly linear and therefore even simple homotopy methods (like fixed step methods) will work.

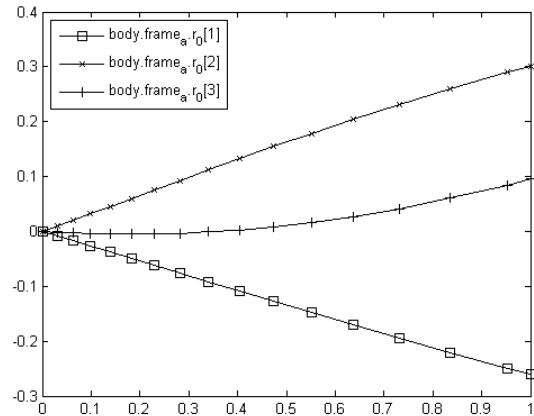


Figure 7: Homotopy path of the absolute position variables for the initialization of Figure 6.

4.2 Analog Electronic Circuit

In electronic circuits, operation starts often after the power supply is switched on. Power supply is in most cases a constant operating voltage of 15V, 5V or others, often a split supply with +15V and -15V is used. After switching on power supply, an initial value of all variables (voltages and currents) is reached, especially capacitors are loaded. The state in which no variable is varying any more is called DC (direct current) operating point. Its calculation is often a challenge for which homotopy operators are useful.

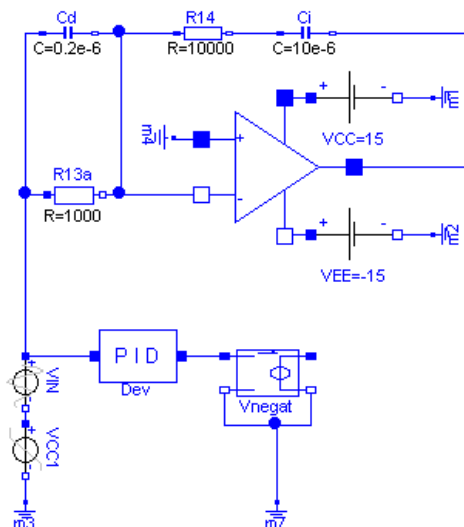


Figure 8: PID controller circuit with $\mu A741$ operational amplifier.

Figure 8 shows a simple PID controller circuit (Tietze and Schenk, 2002) using a μ A741 operational amplifier model (Horowitz and Hill, 1989) which is composed of 21 NPN and PNP transistors of the Modelica Standard Library, see Figure 9:

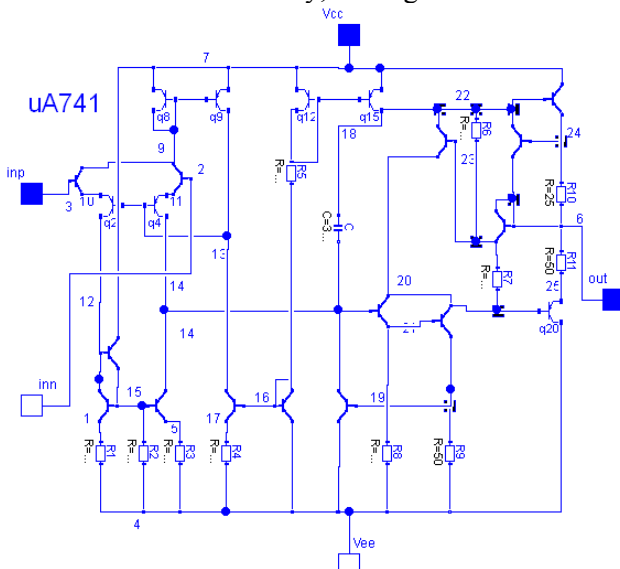


Figure 9: Operational amplifier μ A741 composed of 21 NPN and PNP transistors.

To compare the due power supply (VCC 15V, VEE -15V) limited controller output with the ideal unlimited behaviour, a mathematical PID controller model is inserted in parallel. Typical simulation results with a comparison of the two models are shown in Figure 10. Translating this circuit, results in a system of 240 nonlinear algebraic equations that is reduced by Dymola to a set of 38 nonlinear algebraic equations that have to be solved during initialization (during simulation, only a system of 17 linear equations is present). With Dymola 7.4 (and most likely also with any other Modelica tool), initialization of this circuit fails, i.e., the DC operating point cannot be calculated.

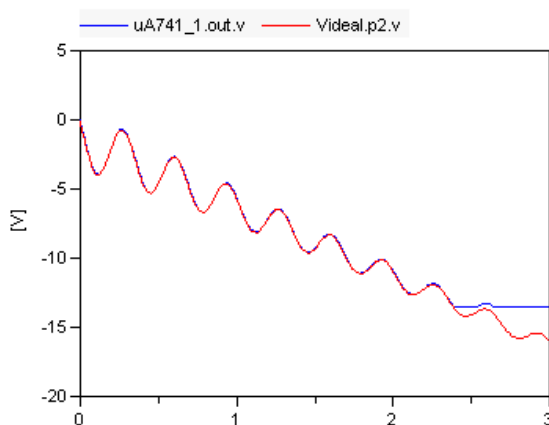


Figure 10: Comparison of circuit PID with the ideal mathematical PID controller.

A successful initialization is possible by replacing the constant supply sources VCC and VEE by ramp sources which start at zero, followed by a transient simulation until all variables remain constant. In general, this way is cumbersome and error prone since the circuit has to be changed manually. Furthermore, the ramping up during a simulation introduces oscillations and simulation has to be long enough until the vibrations “died out”.

The situation changes completely, if the homotopy operator is used by changing the constant voltage model according to

```

model ConstantVoltage_Homotopy
  import Modelica.Electrical.Analog;
  extends Analog.Interfaces.OnePort;
  parameter Modelica.SIunits.Voltage V;
  equation
    v = homotopy(V,0.0);
  end ConstantVoltage_Homotopy;
    
```

This definition starts the constant voltage at zero and during homotopy initialization it is ramped up to the desired voltage V . During the ramping, all derivatives are zero and therefore it is a ramping along steady-states. Simple homotopy algorithms fail in this case. In this example, the Loca algorithm was used to calculate the homotopy initialization. In Figure 11 the non-trivial variation of an internal voltage of the operational amplifier is shown with respect to the homotopy variable λ changing from zero to one. Due to the sharp edge at $\lambda = 0.18$, a homotopy method with a variable step size is needed in this case.

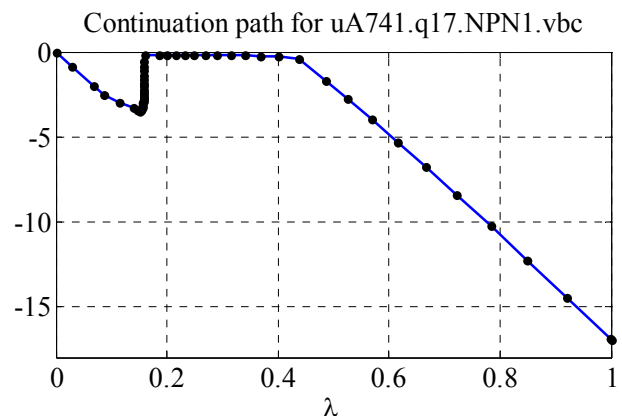


Figure 11: Homotopy path for a voltage variable of the operational amplifier with respect to λ .

4.3 Hydraulic Networks

Hydraulic networks are typically characterized by the simultaneous presence of components with large and small pressure losses, by mixing points, and by nonlinear momentum balance equations, which depend on the fluid properties, e.g., the density. As a

result, the system of nonlinear equations during steady-state initialization is typically large and strongly nonlinear. Their numerical solution is therefore problematic, unless relatively accurate start values are set for the iteration variables.

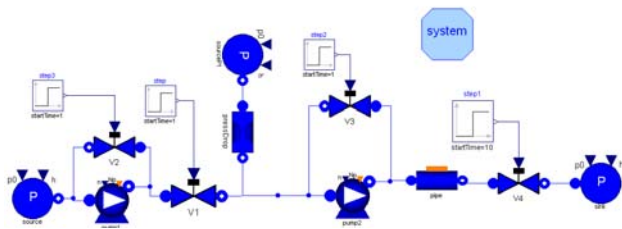


Figure 12: A hydraulic network.

An example case built with the ThermoPower 3 library is shown in Figure 12. A pump with a recirculation and a control valve sucks fluid from a low pressure source. The fluid is then mixed with the flow coming from a second intermediate pressure source through a short pipe, and further pumped through a long pipe (with mass and energy storage) into a high-pressure sink. The pressure losses in the two pipes are small, compared to the pressure losses across the valves and pumps.

The resulting initialization problem has 13 iteration variables after tearing, among which two flow rates and four pressures. If the start values of those six variables are not accurately set, the standard nonlinear solver in Dymola fails to converge.

This initialization problem can be made much easier to solve by substituting the original momentum balance equations in the pump and pipe models by linear, constant-coefficient ones, which are tuned based on nominal operating data, and then by applying the homotopy transformation to bring the model back to its original form.

More specifically, the pressure losses in the short and long pipes are computed by linear m_{flow} - dp relationships, passing through the origin and through the nominal flow and nominal pressure loss point (these data must be provided as parameters). In the case of the pump, the tangent to the flow-head curve at the nominal flow rate is used instead of the original curve. By the simple substitutions of these two equations, the hydraulic problem becomes linear (two linear systems with five and three unknowns), while all the enthalpies and fluid properties are calculated by simple assignments once the flow rates are known. As a consequence, *no start value at all* is required to guarantee convergence of the simplified problem; the homotopy transformation then solves the original nonlinear problem without further intervention by the end user.

It is interesting to note that the homotopy paths of the iteration variables are smooth and do not show

any kind of singularity or turning point even if the actual steady state has a substantial mismatch with the nominal data used to set up the simplified model. As an example, Figure 13 shows the continuation paths for two pressures and two flows if the valve V4 on the far right is closed by 90% at initialization, thus reducing all flows in the circuit to a small fraction of the nominal flow.

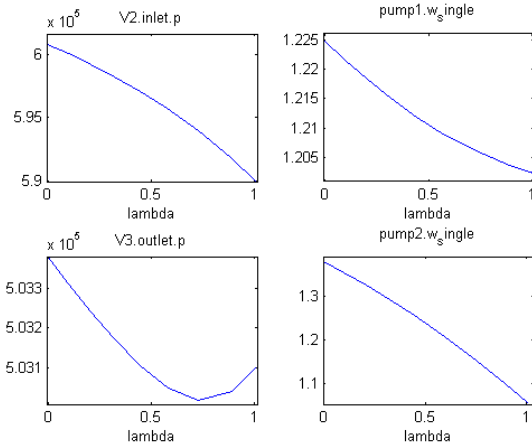


Figure 13: Homotopy paths of 4 iteration variables.

4.4 Calibration of A/C Heat Exchanger

A typical problem in air conditioning system and component design is to calibrate a heat exchanger model to measurement data. This is performed using steady-state initialization in a test bench with given boundary conditions, like the one shown in Figure 14.

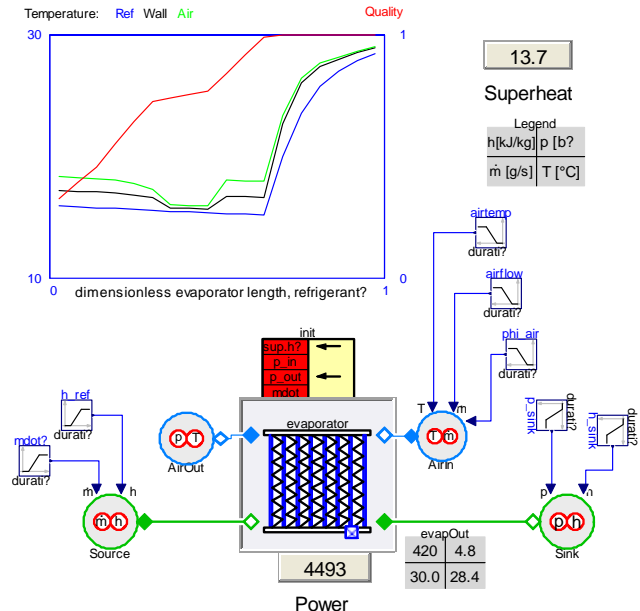


Figure 14: Evaporator calibration test bench, from the AirConditioning library.

Heat transfer on the air side can be correlated using the Nusselt number, $Nu = k_c D_{hyd} / \lambda_F$, which relates the heat transfer coefficient k_c to the hydraulic di-

ameter D_{hyd} and the fluid thermal conductivity λ_f . Normally, the calibration can be performed during initialization by solving for Nu as an unknown parameter using initial equations.

```
parameter Modelica.SIunits.NusseltNumber
  Nu_air(fixed=false, start=10)
  "global Nusselt number";
initial equation
  hex.summary.Qdot_air = P_measured;
```

In most cases this solves perfectly fine using standard methods, and can be combined to calibrate several parameters simultaneously, for example both heat transfer and pressure drop. But sometimes it is difficult to reach the desired solution, P_measured, because it is close to the maximum cooling capacity. The solver will then fail to converge.

Using the homotopy approach, the Nu-number may be used as a control signal, starting at a given value for which the steady-state initialization converges, see the code below:

```
parameter Modelica.SIunits.NusseltNumber
  Nu_air(fixed=false, start=10)
  "global Nusselt number";
parameter Modelica.SIunits.NusseltNumber
  Nu_start=10 "starting Nusselt number";
initial equation
  0 = homotopy(
  actual = hex.summary.Qdot_air - P_measured,
  simplified = Nu_air - Nu_start);
```

The path that the homotopy solver takes can be illustrated with a plot of Qdot_air vs. Nu, see Figure 15. The starting value is taken in the middle of the sloping curve, and the solver will then converge to the desired solution, if one exists. This method has been used to calibrate over large sets of data with excellent results.

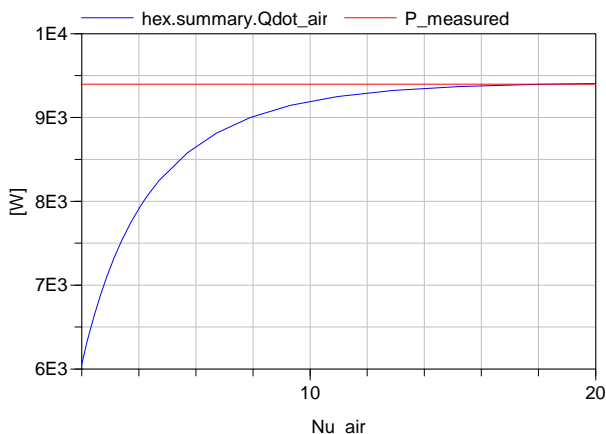


Figure 15: Steady-state performance of heat exchanger as function of Nu-number on the air side. The solution Nu_{air} = 19.7 is very close to maximum Qdot=9420 W.

5 Ill-posed Examples

Unfortunately, it is quite easy to formulate ill-posed problems with the homotopy operator, so that initialization will fail. Below, a number of simple examples are given to demonstrate different kinds of issues.

5.1 Singular Simplified System

The “simplified” problem in the homotopy formulation might be formulated too simple by removing all dependencies of a variable, as shown in the next example:

$$\begin{aligned}x + 2 \cdot \text{homotopy}(y, 1) &= 5 \\ 2 \cdot x - \text{homotopy}(y, 1) &= 0\end{aligned}$$

Note, the “simplified” problem is actually:

$$\begin{aligned}x + 2 &= 5 \\ 2 \cdot x - 1 &= 0\end{aligned}$$

and this equation system does not have a solution although the “actual” problem has a solution. There are different variants of this type of problem. For example, the “simplified” system might remove variables that are used as iteration variables in a system of equations and then the system is singular, although a different selection of iteration variables might make the system regular.

Since such cases can easily appear, the minimum requirement is that a tool reports these problems during translation. Conceptually this is easy, by performing an assignment for the “simplified” problem which would fail (with good diagnostics), if this problem is structurally singular.

A tool might also perform a more involved treatment:

1. For the tearing algorithm, select only iteration variables, that are appearing in the “actual” and in the “simplified” problem formulation (does not work for the problem above).
2. Solve simplified problem with symbolic manipulations (does not work for the problem above).
3. Remove the homotopy operator from certain equations, until the “simplified” system is structurally regular. This would work in the example above, e.g., by removing the homotopy operator from the second equation.
4. The homotopy formulation of appropriate equations is changed. In the example above, one can observe that the modeler defined with the second equation that “y” shall be used for the “actual” problem and “1” for the simplified” problem, i.e., the modeler defined “y=1” for the “simplified” problem. This information

allows to rewrite the second equation to:
 $\text{homotopy}(2 \cdot x, 1) - y = 0$
 which results in a regular “simplified” system.

5.2 Singular Intermediate System

Singular systems might also occur for a combination of the “simplified” and “actual” problem formulation, i.e., when $0 < \lambda < 1$. A typical example is the following where homotopy moves from an “initial state” to a “steady state” formulation (i.e., using Fixed Point Homotopy):

```

model DoNotUse
  Real x;
  parameter Real x0 = 0;
  equation
    der(x) = 1-x;
  initial equation
    0 = homotopy(der(x), x - x0);
end DoNotUse;
    
```

After the initial equation is expanded to

$$0 = \lambda \cdot \dot{x} + (1 - \lambda) \cdot (x - x_0)$$

the two equations can be solved for the unknown x by eliminating the derivative of x :

$$x = \frac{(1 + x_0)\lambda - x_0}{2\lambda - 1}$$

This equation has a singularity at $\lambda = 0.5$, see Figure 16. A homotopy solver will usually not be able to compute the solution and therefore initialization will fail.

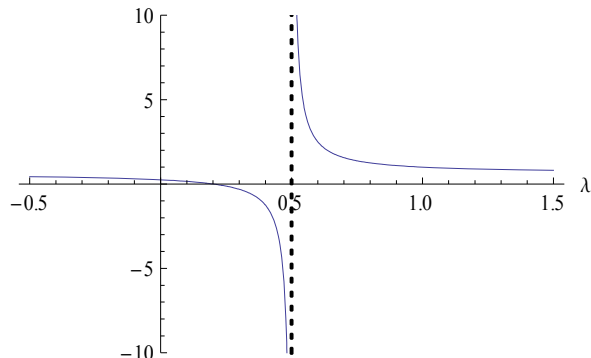


Figure 16: Solution to ill-posed example for $x_0 = 0.25$

5.3 Bifurcation of Intermediate System

Ramping of boundary conditions is a straightforward way to employ homotopy. Some care has to be taken however when using this pattern, which is illustrated for a flip flop, see the simple analog electric circuit of Figure 17:

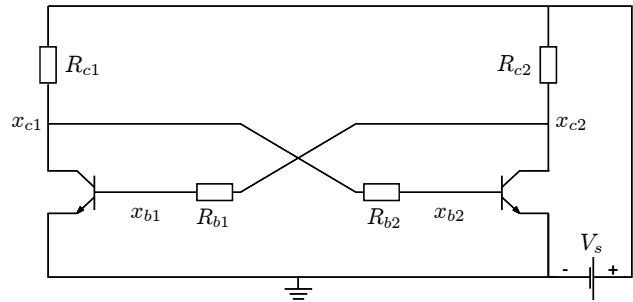


Figure 17: Flip-flop circuit leading to several solutions during the homotopy iteration.

This flip flop circuit has three steady state solutions out of which two are stable. If a homotopy is constructed by ramping up the source voltage V_s , then a bifurcation will show up in the homotopy track. This bifurcation shows up at the point at which the base-emitter junction of the transistor is triggered and the three steady state solutions emerge. In non-trivial applications, such bifurcations are numerically difficult to handle and shall thus be avoided under any circumstances. The following figure illustrates the homotopy trace that results in such a natural parameter continuation strategy (also called source stepping). Here, a simple Ebers-Moll transistor model was used.

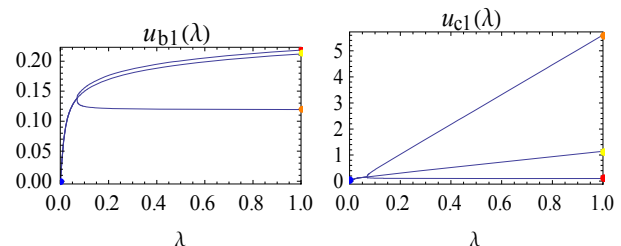


Figure 18: Voltages at base and collector of transistor 1 in flip-flop circuit. At $\lambda = 0.15$ a bifurcation to three solutions occurs.

6 Conclusions

The homotopy operator introduced in to the Modelica language in version 3.2 (*Modelica 2010*) opens up completely new possibilities to robustly initialize Modelica models. Several examples have been given to demonstrate the usage in different domains. Additionally, large power plant applications with up to 671 iteration variables for steady-state initialization are discussed in (*Casella et. al. 2011*). Due to the successful applications, it is planned to introduce this operator at appropriate places in to the next version of the Modelica Standard Library, in order to improve the initialization of Modelica user models.

As demonstrated by several examples in section 5, it is easy to misuse the homotopy operator resulting in failed initialization. As a “rule of thumb”, the

homotopy formulation should not change the “structure” of the equation system, i.e., it should be based on the simplification of terms, but not by solving a completely different problem (e.g. moving from a simplified system that is initialized at given states to a steady-state formulation might easily fail, see section 5.2). Furthermore, it is always useful to inspect how the start-up of the “real” system works and mimic this “start-up” with the homotopy formulation, if this is possible.

There is still room for improving initialization. One issue is that still guess values might be needed for iteration variables (see, e.g., the Delta robot in section 4.1) and the iteration variables are selected by the tool. One remedy might be to introduce an additional enumeration attribute for variables, such as, “iterationSelect” that allows a library developer to directly suggest useful iteration variables with the enumeration values “never, avoid, default, prefer, always”, in a similar way as for the existing attribute “stateSelect” to guide the state selection.

7 Acknowledgements

Partial financial support of DLR and Fraunhofer by BMBF (Förderkennzeichen: 01IS07022F) for this work within the ITEA2 project EUROSYSLIB (www.eurosyslib.com; funding number 06020) is highly appreciated.

References

- Allgower E.L., Georg K. (2003): **Introduction to numerical continuation methods**. SIAM Classics in Applied Mathematics.
- Casella F., Sielemann M., Savoldelli L. (2011): **Steady-state initialization of object-oriented thermo-fluid models by homotopy methods**. Modelica’2011 Conference, Dresden, March 20-22.
- Choi S.H., Book N.L. (1991): **Unreachable roots for global homotopy continuation methods**. AIChE Journal 37, pp. 1093-1095.
- Chow S.N., Mallet-Paret J., Yorke J.A. (1978): **Finding Zeroes of Maps: Homotopy Methods That are Constructive With Probability One**. Mathematics of Computation 32, pp. 887-899.
- Clavel R. (1990): **Device for the Movement and Positioning of an Element in Space**. US Patent No. 4,976,582, December 11, 1990. Download: <http://v3.espacenet.com/publicationDetails/biblio?CC=US&NR=4976582&KC=&FT=E>
- Elib (2010): <http://elib.zib.de/pub/elib/codelib/alcon2/>. Accessed November 2010.
- Dennis J.E., Schnabel R.B. (1996): **Numerical methods for unconstrained optimization and nonlinear equations**. SIAM Classics in Applied Mathematics.
- Deuffhard P., Fiedler B., Kunkel P. (1987): **Efficient numerical path following beyond critical points**. SIAM Journal on Numerical Analysis, Society for Industrial and Applied Mathematics, 24, 912-927.
- Deuffhard P. (2004): **Newton Methods for Nonlinear Problems**. Affine Invariance and Adaptive Algorithms. Springer.
- Dymola (2010): **Dymola 7.4**. <http://www.3ds.com/products/catia/portfolio/dymola>
- Heroux M.A., Bartlett R.A., Howle V.E., Hoekstra R.J., Hu J.J., Kolda T.G., Lehoucq R.B., Long K.R., Pawlowski R.P., Phipps E.T., Salinger A.G., Thornquist H.K., Tuminaro R.S., Willenbring J.M., Williams A., Stanley K.S. (2005): **An overview of the Trilinos project**. ACM Transactions on Mathematical Software, 31, 397-423.
- Hompack (2010): <http://www.netlib.org/hompack/>. Accessed November 2010.
- Horowitz P., Hill W. (1989): **The Art of Electronics**. Cambridge University Press, page 189.
- Keller H. (1978): **Global homotopies and Newton methods**. C. de Boor and G. Golub, eds., Academic Press, New York, pp. 73-94.
- Kelley C.T. (2003): **Solving nonlinear equations with Newton's method**. SIAM.
- Mattsson S.E., Elmqvist H., Otter M., Olsson H. (2002): **Initialization of Hybrid Differential-Algebraic Equations in Modelica 2.0**. Proceedings of the Second International Modelica Conference, Munich, Germany, pp. 9-15. Download: https://www.modelica.org/events/Conference2002/papers/p02_Mattsson.pdf
- Modelica (2010): **Modelica – A Unified Object-Oriented Language for Physical Systems Modeling. Language Specification, Version 3.2**. March 24. Download: <https://www.modelica.org/documents/ModelicaSpec32.pdf>
- Tietze U., Schenk C. (2002): **Halbleiterschaltungstechnik**. Springer, 12th edition, page 1150.
- Watson L.T., Billups S.C., Morgan A. P. (1987): **Algorithm 652: HOMPACT, A suite of codes for globally convergent homotopy algorithms**. ACM Transactions on Mathematical Software, 13, 281-310.
- Wayburn T., Seader J. (1987): **Homotopy continuation methods for computer-aided process design**. Computers & Chemical Engineering 11, pp. 7-25.

Steady-state initialization of object-oriented thermo-fluid models by homotopy methods

Francesco Casella* Michael Sielemann† Luca Savoldelli*

* Dipartimento di Elettronica e Informazione, Politecnico di Milano
Piazza Leonardo da Vinci, 32, 20133 Milano, Italy

† Deutsches Zentrum für Luft- und Raumfahrt, Institute of Robotics and Mechatronics
Münchner Strasse 20, 82234 Wessling, Germany

Abstract

The steady-state initialization of large object-oriented thermo-hydraulic networks is a difficult problem, because of the sensitivity of the convergence to the initial guesses of the iteration variables. This paper proposes an approach to this problem based on homotopy transformation, detailing specific criteria for model simplifications in this physical domain. The approach is successfully demonstrated on large power plant test cases, having several hundreds of iteration variables.

Keywords: Thermo-hydraulic systems, power plants, steady-state initialization problems.

1 Introduction

Steady state initialization of large thermo-fluid network is hard and often fails, even when using state-of-the-art nonlinear solvers. This hampers the use of object-oriented models in applications such as power plant simulation, because of the difficulties encountered in getting a newly built model to actually simulate.

Currently, the only way to solve this problem is to manually set good initial guesses for all the iteration variables of the problem. This is rather inconvenient, since the number of such variables can easily grow beyond a hundred or even a thousand, and also because any tiny change to the model, or to the version of the Modelica tool used to simulate it, can lead to a different set of iteration variables and thus require a further setting of initial guesses. This makes the initialization activity tedious and very far from the concepts of modularity and object-orientation.

This paper presents an alternative approach to the problem, based on homotopy transformation. The proposed strategy is demonstrated by means of a proto-

type solver code on large-scale power plant test cases.

The paper is structured as follows: Section 2 gives the basic of homotopy-based initialization of object-oriented models and presents the test implementation of the solver. Section 3 introduces criteria for the formulation of simplified models in the domain of thermo-hydraulic networks. Section 4 illustrates experimental results obtained large-scale models of combined-cycle power plants, while Section 5 gives concluding remarks.

2 Homotopy-based initialization of object-oriented models

2.1 Problem definition

To encode initialization problems in Modelica, language constructs such as initial equation sections are defined. They introduce additional constraints, which, together with all equations and algorithms that are utilized during simulation, constitute the initialization problem. The solution can then be used to assign all variables, derivatives and pre-variables consistent values.

Formally, the resulting problem is an initial value problem for a system of differential algebraic equations (DAE), $0 = F(\dot{x}, x, w, t)$. Variables x are the state variables, w are the algebraic unknowns, and t is time. The initialisation problem prescribed by the model introduces conditions such as the steady-state condition $\dot{x} = 0$ at some time $t = t_0$. The differential algebraic equation system is usually index reduced, i.e. it has index 1, which means that the following expression be regular

$$\begin{bmatrix} \frac{\partial F}{\partial \dot{x}} & \frac{\partial F}{\partial w} \end{bmatrix}.$$

Formally, this problem usually results in a nonlinear system of algebraic equations that has to be

solved numerically. Unfortunately, this does not always work robustly for industrial problems as the frequently utilised gradient-based local algorithms such as damped Newton Method offer local convergence properties only (even when using so-called globalizations such as trust regions).

Several alternative methods are discussed in literature to solve the present problem more robustly. Homotopy continuation is one of them and it is considered in this article to address the need for more robust initialisation.

2.2 Established homotopy methods

Informally, using homotopy to solve nonlinear algebraic equation systems can be defined as follows. First, one starts with a simple problem whose solution is known or easy to obtain and then continuously deforms this simple problem to the difficult problem of interest. Conceptually, this appears to be simple. However, several details of these methods have to be taken into account. In particular, the existence of the homotopy path between the start and a solution, finite length, and nonexistence of singularities along the track are not guaranteed.

In order to construct a homotopy, one needs the system of residual equations of interest, $F(x)$, and another one that is easy to solve $\tilde{F}(x)$. Here and in the remainder of this section, a generic vector of unknowns x is indicating, including the state derivatives, states, and algebraic unknowns. The two sets of residual equations are then deformed from one to the other via a homotopy or continuation parameter λ . A simple example of such a deformation is a linear convex combination. In any case, the homotopy is then a system of equations with one higher dimension and denoted by

$$\rho(x, \lambda) = 0.$$

The homotopy parameter is typically restricted to some range, e.g. $[0, 1]$, such that $\rho(x, 0) = \tilde{F}(x) = 0$ is solved easily and $\rho(x, 1) = F(x) = 0$ is the system of interest.

Many general-purpose homotopies are defined in literature. For example, the Newton homotopy [3] is defined as:

$$\rho(x, \lambda) = \lambda F(x) + (1 - \lambda)(x - x_0), \quad (1)$$

where x_0 is a tentative estimate for the solution of $F(x) = 0$. Other similar methods exist, such as the fixed point homotopy and the affine homotopy. All such methods exhibit convergence failure modes, as

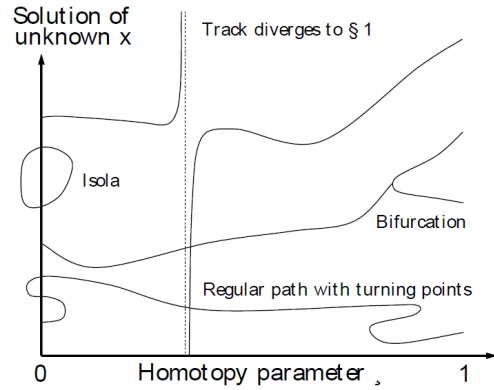


Figure 1: Problematic homotopy paths

shown in Fig. 1, which render them not sufficiently robust to alleviate the convergence issues described in the introduction. Examples of these convergence failure modes are infinite loops without reaching $\lambda = 1$ (isolae), which occur for the Newton homotopy, and components of the solution vector wandering off toward $\pm\infty$, as observed for the fixed point and affine homotopies. Furthermore, bifurcations may arise along the continuation paths, which are non-trivial to handle numerically for industrial problems.

Additional reasons why established homotopy methods are not considered a feasible solution to alleviate the need for a more robust initialisation method are given in [1].

2.3 Problem-specific homotopy

In the established homotopies mentioned in the previous section, two rather unrelated systems of equations are continuously deformed into each other; the radical difference between the two systems of equation is arguably the cause of the singular homotopy paths.

Therefore, we propose to introduce problem-specific homotopies, where the simplified system is derived from the actual system of interest and close enough to it so as to avoid that the homotopy to the actual problem of interest be free of singularities. The formulation of the simplified systems is introduced by domain experts, allowing them to infuse their knowledge about the physics of the problem into the way the equation system is solved. The approach is fully compatible with object-orientation and declarative modeling.

2.4 Test implementation

In order to validate the methodology a test implementation was developed. It was based on the equation-

based object-oriented modelling language Modelica[®] and the compiler Dymola[®] in versions 7.3 and 6.1.

Using this test implementation, homotopies such as

$$\rho(x, \lambda) = \lambda F(x) + (1 - \lambda) \tilde{F}(x)$$

can be formulated in a declarative way, where $F(x)$ is the actual problem and $\tilde{F}(x)$ is the simplified one. For this purpose, a function `homotopy()` was introduced, that takes two input arguments, namely `actual`, the expression describing the actual problem, and `simplified`, the expression corresponding to the simple problem. The Modelica compiler then expanded this function according to the above-described homotopy. For example, the expression

```
homotopy(actual=a*b, simplified=c/d)
```

was expanded to

$$\lambda(a \cdot b) + (1 - \lambda)(c/d).$$

This idea has later on been included in version 3.2 of the Modelica language specification, where a built-in `homotopy()` operator with the same semantics has been introduced.

From the numerical side, the test implementation utilised the LOCA continuation algorithms of Trilinos [2]. A list of the main features of the test implementation is given here.

First of all, the algorithm provided three options for the implementation of the `homotopy()` function. In order to numerically solve the simplified problem as easily as possible, a version of the function that returned the `simplified` argument was inlined, in order to obtain the maximum structural simplification of the corresponding system of equations. For the homotopy transformation, it was expanded to the given homotopy expression. For the dynamic simulation of the system, after initialization, an inlined version returning the `actual` argument was used.

The user was able to manually prescribe whether to use homotopy initialisation or not. This is an important feature for library development and debugging, and may be useful for end users, too (e.g., if a local gradient based solver converges to a mathematically valid, but physically unreasonable solution or when a local gradient based solver does not converge and a user does not want to wait at the start of each simulation until the software realised this).

The user was able to specify that the simplified problem only should be solved. This feature is essential for library development, when one must analyze

the properties of the simplified model, to understand whether it is good enough to provide a robust numerical initialization to the homotopy transformation, or if it is necessary to proceed further in the simplification process.

Verbose information on the homotopy was optionally provided, which was useful for library development and debugging, and the homotopy traces of all the iteration variables of the nonlinear system of equations were recorded for later visualisation and analysis.

Last, but not least, the underlying solver was able to follow homotopy traces with turning points, should they arise during the transformation.

3 Homotopy-based initialization of thermo-fluid network models

3.1 Basic principles

The basic idea is to formulate a simplified model which is easier to solve without the need of accurate start values, but which is on the other hand close enough to the actual problem to avoid singularities during the homotopy transformation. Three goals must be pursued:

1. The simplified model should approximate the actual model around the nominal operating point of the plant, in order to have a solution which is close to that operating point, and thus physically meaningful.
2. The simplified model should be close enough to the actual model that the homotopy transformation from the simplified to the actual problem gives rise to smooth transformations of all the iteration variables, with no singularities, no bifurcations, and possibly no turning points, even though the latter ones can be handled by continuation algorithms such as LOCA.
3. The numerical solution of the simplified model should converge with rough (default, or nominal-parameter based) initial guess values, either set by default or based on parameters specifying the nominal operating point. This avoids the need of manually setting start values for the iteration variables of the specific system at hand, whose set is difficult or impossible to determine a-priori by the end user, as it is usually the result of sophisticated (and often proprietary) tearing algorithms.

3.2 Formulating the simplified model

The simplified model should be initialized at steady state as the actual problem, in order to avoid unphysical situations, so the initial equations ($\text{der}(x) = 0$) are not changed. The general guideline is to approximate a few model equations so that the implicit system of equations corresponding to the steady-state initialization problem has the minimum number of unknowns and is as linear as possible. Noting that a linear problem can be solved by the standard Newton algorithm exactly in one iteration, one can expect that in general the less nonlinear the problem is, the less sensitive to start values the convergence will be.

Ideally, the simplified model could be obtained by linearizing the actual model close to the initial steady state. In practice, this is not a good idea for two reasons. First of all, this strategy would require to modify a large fraction of the model equations (all the nonlinear ones). Secondly, obtaining such linearisation requires to know the steady state values of all the variables, which are yet to be determined. The idea is then instead to simplify only those few equations that mostly contribute to the nonlinearity of the large implicit system of equations of typical steady-state initialization problems.

Power plant models are essentially thermo-hydraulic networks with non-trivial fluid models (ideal gases with temperature-dependent c_p and vaporizing fluids, usually water), exchanging heat by convection through heat exchanger walls. The main sources of nonlinearities in the steady state initialization problem are now listed.

1. *Momentum balance equations*: pressure-flow rate relationships are usually quadratic and depend on upstream properties, such as density and viscosity, which in turn depend on thermal variables and on the flow direction.
2. *Energy balance equations* have the form $\sum_j w_j h_j + \sum_j Q_j = 0$, thus are nonlinear in the mass flow rate - specific enthalpy products $w_j h_j$.
3. The *upstream enthalpy* appearing in energy balance equations of components allowing flow reversal depends on the direction of the flow.
4. *Flow-dependent heat transfer coefficients* γ introduce nonlinearities in heat transfer equations $Q = \gamma S(T_{\text{fluid}} - T_{\text{wall}})$.
5. *Temperature-enthalpy relationships* are nonlinear in both ideal gas and water/steam models.

6. *Controllers influencing flow rates* through, e.g., valve openings, pump speeds, etc., and whose controlled variables are instead related to energy flows or storage, e.g., turbine power, boiler pressure, introduce nonlinear couplings between hydraulic and thermal equations.

7. *Controllers with control signal saturations* introduce nonlinearities in the system model.

Note that many other nonlinear equations which are present in the model are irrelevant for the steady-state initialization, because they only involve the dynamic behaviour, which is by definition not considered if all derivatives are zero. For example, the dependency on pressure and temperature (or specific enthalpy) of the fluid compressibility $\frac{dp}{dp}$, which enters the left-hand-side of dynamic mass balances, is irrelevant in the determination of the steady state. Therefore, if structural analysis is applied to the initialization problem, the computation of those quantities will be moved after the core implicit system of equations in the BLT transformation, and they will be computed explicitly as a function of the already computed thermodynamic states, e.g., (p, T) or (p, h) . Consequently, it is only necessary to worry about those equations and those variables which are strictly necessary to solve the steady-state equations, where one assumes that all derivatives are equal to zero.

3.2.1 Momentum balance equations

The most important source of nonlinearity in the steady-state initialization problem is given by the momentum equations, which are usually quadratic in the flow rate, due to the friction term. When low pressure losses are modelled, the flow rate is highly sensitive to pressure errors: a small error in the pressures during the first Newton iterations can cause large errors in the flow rates, which in turn cause large errors in the energy balance equations, possibly bringing the specific enthalpies out of their validity range of the fluid model. Furthermore, the dependence of the momentum balance on the fluid properties introduces a nonlinear coupling between the hydraulic equations, describing pressure-flow relationships, and the thermal equations, describing energy storage and heat transfer.

All these problems are removed if the momentum balances are substituted with linear constant-coefficient pressure-flow rate relationships. These can be based on nominal operating data (nominal pressure drop, nominal flow rate), which are often already in-

cluded among the component parameters, and are usually known in advance from overall plant design data. Friction losses can be approximated by a linear function passing through the origin and the nominal operating point:

$$w = \frac{w_{nom}}{\Delta p_{nom}} \Delta p \quad (2)$$

Static head terms can be computed using a constant known nominal density:

$$\Delta p_{static} = \rho_{nom} g H \quad (3)$$

The flow characteristic of turbines can be approximated by a linear pressure-flowrate relationship:

$$w = \frac{w_{nom}}{\Delta p_{nom}} \Delta p \quad (4)$$

Control valves can be represented by a simplified equation, where the flow rate is both proportional to the pressure difference Δp and to the valve opening α :

$$w = \alpha \frac{w_{nom}}{\Delta p_{nom}} \Delta p. \quad (5)$$

This equation is still significantly nonlinear and might cause problems, in particular if α is the output of a controller (e.g., in the case of level controller for drum boilers). In this case, it is possible to further simplify the equation, making it linear, by removing the dependency on Δp :

$$w = \alpha w_{nom}. \quad (6)$$

Pump characteristics cannot be reasonably represented by a curve passing through the origin. In this case, it is convenient to use a linearised version of the characteristic curve, computed around the nominal flow rate, head, and pump rotational speed.

The simplified models are then written together with their actual counterparts, using the `homotopy()` operator. A few examples from the ThermoPower library are shown here for the sake of the example:

```
// Pressure loss component
pin - pout =
  homotopy(smooth(1, Kf*squareReg(w,wnom*wnf))/rho,
            dpnom/wnom*w) "Flow characteristics";

// Valve for incompressible fluid
w = homotopy(FlowChar(theta)*Av*sqrt(rho)*sqrtR(dp),
            theta/thetanom*wnom/dpnom*dp);

// Pump
function df_dq = der(flowCharacteristic, q_flow);
head = homotopy((n/n0)^2*flowChar(q*n0/(n + n_eps)),
                df_dq(q0)*(q - q0)+
                (2/n0*flowChar(q0) - q0/n0*df_dq(q0))*(n - n0)
                + head0);

// Turbine
w = homotopy(Kt*partialArc*sqrt(p_in*rho_in))*
  sqrtReg(1 - (1/PR)^2),
            wnom/pnom*p_in);
```

In some cases, the structure of the system of equations corresponding to the simplified steady-state initialization is such that, with these simplifications, the steady-state hydraulic equations are completely decoupled from the steady-state thermal equations. In those cases, the BLT algorithm will split the system of equations into two smaller subsystems. First, the hydraulic equations alone will be solved, determining the pressures and flow rates. Since all the involved equations are now linear, the problem is solved easily and without any concern about convergence and initial guess values. Subsequently, the thermal equations will be solved, but since the flow rates are now known, a major source of nonlinearity, i.e., the $w_j h_j$ products in energy balances, will be gone, thus making it easier to solve the thermal equations as well.

Other cases will not be this easy. Consider for example a Rankine cycle with a circulation boiler. Even though the simplified flow equation for the turbine is linear, it is apparent how the steam flow rate essentially depends on the heat input, which determines how much steam is produced. Consequently, the hydraulic and thermal equations will be coupled in this case, even when considering the simplified model. Anyway, a larger part of the equations in this system will be linear, thus easing the convergence of the nonlinear solver. More opportunities for efficient tearing will also be available, since a larger fraction of equations can be symbolically turned into an explicit assignment.

3.2.2 Energy balance equations

The nonlinearity in this case stems from the $w_j h_j$ products in the steady-state energy balances. It is not as hard as in the case of momentum balances for small pressure losses, but it can still give rise to significant problems: if during iterations, the mass flow rate is wrong by a factor of, say, two, then also enthalpy changes will be off by the same factor, which could cause out-of-bounds problems with the fluid property computation routines.

The best way to get rid of this problem is to use the nominal flow rates instead of the actual flow rates for the simplified initialization problem; by doing so, the energy balance equations become linear, and are thus solved without major problems. Unfortunately, specifying all the nominal flow rates is rather impractical for multiple-port mixing components such as storage and storage-less mixing volumes, steam drums, steam headers, etc. Furthermore, if all of those nominal values were not set to correct values, considerable errors could arise in the computation of the enthalpies, that

could hamper the convergence of the simplified problem.

A reasonable compromise, which allows to get rid of most w_j/h_j -type nonlinearities in typical power plant models, is to using the nominal flow rate in the energy balance equations for both sides of heat exchangers, instead of the actual flow rate. The values of the nominal primary and secondary flow rates are usually known for all heat exchangers in a plant, only two parameters are needed per heat exchanger, and a lot of nonlinear equations are turned into linear equations, since there are $2N$ such energy balance equations in a heat exchanger having N discrete volumes on each side, and there are usually many heat exchangers in a given plant model. It is assumed that the few remaining nonlinear energy balance equations (contained in mixers, drums, steam headers, etc.) will be handled by the nonlinear solver without major problems.

Note that this approximation effectively removes the dependency between the flow rate and the outlet temperature of the fluid. This might then prove problematic in all those cases where a temperature controller is used to keep the outlet temperature at a given set point, because the corresponding simplified equations might become singular or ill-conditioned. In those cases, it is necessary to open the temperature feedback loop in the simplified problem - see below Sect. 3.2.6.

3.2.3 Dependence of the upstream enthalpy on the direction of the flow in energy balances

If flow reversal is allowed, the specific enthalpy of fluids entering and leaving control volumes where mass and energy balances are formulated are calculated using the upstream discretisation scheme, e.g.:

$$h = \text{if } w > 0 \text{ then } h_{\text{entering}} \text{ else } h_{\text{internal}} \quad (7)$$

The discontinuity might be smoothed out in the neighbourhood of $w = 0$, but in any case these equations introduce a strong nonlinearity, if not a discontinuity, in the steady-state equations.

If the hydraulic equations of the simplified problem are completely decoupled from the thermal equations, then this is not a problem: the values of all flow rates will be computed by solving the linear hydraulic equations; then, the flow rate w will no longer be an unknown when (7) will be solved. In general, this decoupling cannot be performed, as discussed in the previous sub-section. Upon initialization, however, one can assume that the flow rate will have the design direction, so a simplified equation can be written under

that assumption, e.g.:

$$h = h_{\text{entering}} \quad (8)$$

For example, this is how the specific enthalpy at the inlet port of a mixing volume is computed in the ThermoPower library:

```
hi = homotopy(if not allowFlowReversal
              then inStream(inlet.h_outflow)
              else actualStream(inlet.h_outflow),
              inStream(inlet.h_outflow));
```

3.2.4 Flow-dependent heat transfer coefficients

Convective heat transfer is represented by equations such as

$$Q = \gamma S(T_{\text{fluid}} - T_{\text{wall}}) \quad (9)$$

Simpler models assume a constant heat transfer coefficient γ , so the equation is linear. More accurate models instead compute γ as a function of Reynolds and Prandtl numbers, which depend on the flow rate, as well as on the density, viscosity and thermal conductivity of the fluid, and possibly also on the wall temperature. All these dependencies introduce considerable nonlinearities, as well as coupling between the hydraulic and thermal equations.

The obvious strategy for simplified problem formulation is to use the nominal value of γ instead of the actual one, thus making equation (9) linear, e.g.:

```
wall.gamma[j] = homotopy(
  gamma_nom*noEvent(abs(infl.m_flow/wnom)^kw),
  gamma_nom);
```

3.2.5 Temperature-enthalpy relationships

Temperature profiles and transferred thermal power in heat exchangers are determined by the interplay between heat transfer, which is driven by temperature differences, and convective transport of heat by the fluid, which is described by enthalpy differences. The temperature-enthalpy relationships are therefore involved in the steady-state equations describing heat exchangers, namely $h = h(T)$ for ideal gases and $T = T(p, h)$ for vaporizing fluids.

In the case of ideal gases, the function is approximately linear over significant ranges of T , since its derivative, the specific heat c_p , does not change too much with the temperature. This is also the case for the vaporizing fluid, as long as the function is evaluated on the correct side of the saturation curve: the c_p of liquid water does not change dramatically with temperature, nor does the c_p of steam, with the exception

of the transcritical region and of a thin region just outside the saturation curve. On the other hand, substituting those functions with linear approximations which are consistent with each other is not trivial and would require substantial changes to the code of the original fluid model.

The strategy for the homotopy is then to rely on the fact that these functions are only mildly nonlinear, so they should not cause major convergence issues, of course as long as they are called in their range of validity. Therefore, the corresponding function calls are left untouched in the simplified model.

It is however essential to select reasonable start values for the gas temperatures, so that the guess values used for the first Newton iterations are already in the correct temperature range, as far as c_p is concerned; the precise numerical value of start attribute is not critical. As concerns the vaporizing fluid properties, start values should be selected so that the first Newton iterations compute the properties on the correct side of the saturation curve, i.e., subcooled liquid or superheated steam.

The user input is therefore a very rough temperature value for the gas side (say, 400 rather than 600 or 800 K, for standard flue gas heat exchanger), and the indication of a nominal pressure and of the phase (liquid or vapour) for the vapour size, that can be used internally in the model to compute start values of the specific enthalpies corresponding to well-subcooled liquid and well-superheated steam.

Evaporating pipes are less critical from this point of view, because in a two-phase mixture the temperature-enthalpy relationship becomes flat, i.e. the temperature no longer depends on the enthalpy, but only on the pressure, which usually does not change much across the pipe length.

3.2.6 Controllers acting on flows and controlling energy-related quantities

It is often the case that the plant model is complete with controllers, and that the goal is to initialize the whole controlled system in steady state. If the controller is active and contains some integral action on the error, the steady-state equations are equivalent to the equation

$$y = y_{sp}, \tag{10}$$

where y_{sp} is the value of the set point. This equation, coupled with the rest of the plant model, implicitly determines the value of the control value, e.g. a valve opening or a pump rotational speed.

If the control variable directly influences a flow rate, and the controlled variable is mainly determined by the energy flows, (10) introduces a strong nonlinear coupling between the hydraulic equations and the thermal equations, thus hampering the solver convergence.

Consider the following example. The last economizer stages of a heat recovery steam generator usually allow to modulate a recirculation flow in order to control the outlet temperature of the preheated water to the desired value. In order to change this value, valves or pump speeds must be changed, that can also affect the water/steam flow through the evaporator and superheater, thus greatly influencing all the thermal power transfer phenomena across the steam generator. During the first iterations of the nonlinear solver, the gas temperature near the exhaust can be quite different from its design value: this causes the recirculation flows to be also different from the design values, thus influencing the evaporator and superheater flows, which in turn affect the temperature of the gas heating them. In the end, the solver might get stuck far away from the sought after solution even when considering the simplified equations for the physical model.

Should this happen, it usually is possible to roughly estimate what the value of the control variable will be in the nominal operating point of the plant. It is then possible to remove the above-described nonlinear coupling by using a simplified model of the controller that just outputs the start value of the control variable. Of course this means that the steady-state operating point of the simplified model will be slightly off with respect to the correct value, but this is not a problem, as long as the operating point is physically meaningful and not too far from the exact solution. The homotopy transformation will then slowly introduce the closed-loop controller action, thus smoothly bringing the controlled variables to their set points at the end of the homotopy transformation.

In some cases, an explicit controller model is not included in the plant model, and the steady-state operating point is just obtained by adding equations such as (10) for the desired outputs (inverse initialization). In this case, those equations should use the homotopy operator to blend the prescribed control value (simplified model) with the prescribed output value, e.g.:

$$0 = \text{homotopy}(\text{valve_opening} - \text{valve_opening_nominal}, \text{T_out} - \text{T_out_nominal});$$

3.2.7 Controllers with control signal saturations

It is often the case that controllers in controlled plant models include saturations, i.e., limitations in the control variable range. If the saturation limits are wide

enough, then they are actually irrelevant: the controller is modulating, and the effect of the controller is equivalent to the steady-state equation on the integral action. This is in turn equivalent to (10), which implicitly determines the value of the control variable, within the saturation limits. On the other hand, if an out-of-bound control action would be required to attain the set point, then the saturation is engaged, equation (10) no longer holds and is replaced by either

$$u = u_{max} \quad (11)$$

or

$$u = u_{min}. \quad (12)$$

Irrespective of the way the saturating controller is actually implemented (e.g. with or without anti-windup action), the above scenarios always hold, indicating a strongly nonlinear behaviour of the corresponding system of equations. In other words, letting the solver figure out which controllers are in a modulating state, which are saturated high and which are saturated low corresponds to solving a highly nonlinear problem, with potentially combinatorial complexity, which can cause serious convergence problems to the solver.

Doing so is however not necessary in general, since the status of the controllers in the nominal operating point is usually well known. In case it is known in advance that the controller will be modulating, then the saturation limits can be removed in the simplified model, thus making the model linear. In case it is known in advance that the control output will be saturated at the maximum or minimum limits, then the saturation equation is replaced with an equation stating that the control output is fixed at the maximum or minimum value.

Note that this functionality can be merged with the functionality described in the previous subsection. Summing up, the simplified model should either remove the saturation limits from the output, or hold the output at a fixed value, which might be a specific nominal value, the maximum, or the minimum, depending on the situation.

3.3 Solving the simplified model

It is apparent that all the above-described simplification strategies reduce the couplings between equations and the nonlinear effects, compared to the actual initialization problem. In order to take full advantage of these simplifications and ensure the highest chance of convergence, it is recommended that the tool applies

structural analysis and optimization (BLT transformation, tearing, etc.) to the simplified initialization problem, obtained by replacing all instances of the homotopy() operator with their simplified argument.

When this is done, then the iteration variables of the initialization problem, i.e. the tearing variables, typically belong to these categories:

- Gas-side temperature distributions in heat exchangers
- Wall temperatures distributions in heat exchangers
- Water/steam side enthalpy distributions in heat exchangers
- Steam drum pressures
- A few other flow rates and pressures

The first two sets will need very rough start values (say 400, 600 or 800 K, depending on the heat exchanger); there will be no need at all to provide estimates of the actual temperature distributions within heat exchangers. The third set also requires very rough start values (subcooled liquid or superheated steam, depending on the case). Therefore, appropriate start values can be set for all these variables in the model, based on a couple of numerical parameters in the heat exchanger component, whose precise value is not at all critical for convergence. Steam drum start values can be easily supplied based on nominal operating point data. If the flow rates belong to heat exchanger components, a nominal value is already available, since it is required to perform the energy balance equation simplification, so it can also be used to set the start value.

In some cases, there might still be a very few remaining iteration variables that don't have any meaningful start value, causing the solver to fail. These can be fixed on a case by case basis, or by adding suitable start and/or nominal parameters to the corresponding library model. Ideally, required start values should be inferred from parameters of the component which give information about the nominal operating point, without the need of extra ad-hoc input by the end-user.

3.4 Steady-state initialization far from the nominal operating point

The simplified problem has been designed to approximate the actual problem at the nominal operating point. What if one wants to initialize the plant at a

different operating point, e.g. 40% load, instead of the nominal 100% load?

One idea in this case is to use the `homotopy()` operator to parameterize the signal sources that define the operating point. For example, if the load set-point is generated by a step or ramp source, one might write `homotopy(40, 100)` as the offset value. This means that the simplified initialization problem is actually solved with an offset of 100%, i.e., at full load; during the homotopy transformation, while the problem is brought to its actual form, the load is also progressively reduced to 40%, thus eventually converging to the required steady state.

It has been verified in a number of test cases (see next Section) that this additional mismatch between simplified and actual model does not lead to any singularity of the solution during the homotopy transformation, and guarantees successful convergence of the actual initialization problem.

Should this not be the case, two homotopy transformations should be performed in sequence: first the simplified problem at nominal load should be transformed to the actual model, also at nominal load; then, the load set point should be reduced, therefore realising a quasi-static change of the operating point from full load to partial load, which should pose no problems. Unfortunately this is not possible with the current definition of the `homotopy()` operator, which only allows for a single, system-wide transformation.

4 Experimental results

The general ideas illustrated in the previous sections has been implemented in the version 3 of the `ThermoPower` library [4]. The library has then been used to build a series of test cases of increasing complexity, culminating in the complete model of a combined-cycle power plant, whose heat recovery steam generator (HRSG) includes 15 different heat exchangers. A few extra parameters for nominal values required by the simplified model had to be added to the formerly developed heat exchanger models; however, they are a very small fraction of the number of parameters already needed to set up those models and, as noted in the previous section, their numerical values need not be precise by any means. Furthermore, and more important, these parameters are set once and for all in a given plant model and need not be changed on a case-by-case basis depending on the choice of start values of the Modelica tool.

For the simpler cases, homotopy was actually not

necessary to solve the initialization problem, but as the complexity increased, more and more cases fail to initialize when the built-in solver of Dymola is used, because of initial guesses which are not accurate enough. All the simplified models converged without problems (as long as the nominal parameter gives a correct order of magnitude for all the iteration variables) and the homotopy paths of all the iteration variables proved to be smooth and devoid of turning points or worse singularities.

The three largest and hardest-to-solve cases are briefly documented here. The model describes a complete combined-cycle power plant. The three levels of pressure HRSG includes 15 heat exchangers, each one discretized by finite volumes and with flow-dependent heat transfer coefficients. The steam turbine system includes a condenser model and a pumping system model, so the water/steam cycle is closed. The turbines operate in sliding pressure; control loops are included to control the steam drum levels, the superheater outlet temperatures, the economizer outlet temperature, and the combined electrical power output of the gas and steam turbines. Three variants have been considered:

1. reference plant model, initialized at 100% load;
2. reference plant model, initialized at 60% load;
3. detailed plant model, with two parallel HRSGs, common steam collector and steam turbine system, also initialized at 100% load;

The initialization problem of case 1. has 345 iteration variables. During the homotopy transformation, no variable shows bifurcations or turning points. Most variables change by less than 5% between $\lambda = 0$ and $\lambda = 1$. The outputs of the superheaters and reheaters temperature controllers (which are fixed to the start value at $\lambda = 0$ and work in closed loop at $\lambda = 1$) show the biggest variations, but change smoothly and without singularities during the homotopy transformation.

The computation of the transformation took 8 steps and 40 seconds using the test implementation, running on a 2.26 GHz P9300 Intel processor. For the sake of the example, Fig. 2 shows some representative plots of iteration variables during the transformation.

Case 2 has the same number of iteration variables, but now the homotopy transformation also involves bringing down the load from the nominal 100% value to 60%, so it is a bit more involved, because the values of the initial steady state significantly differ from the nominal values. This time, the transformation required 37 steps and took 100 seconds to compute. Fig.

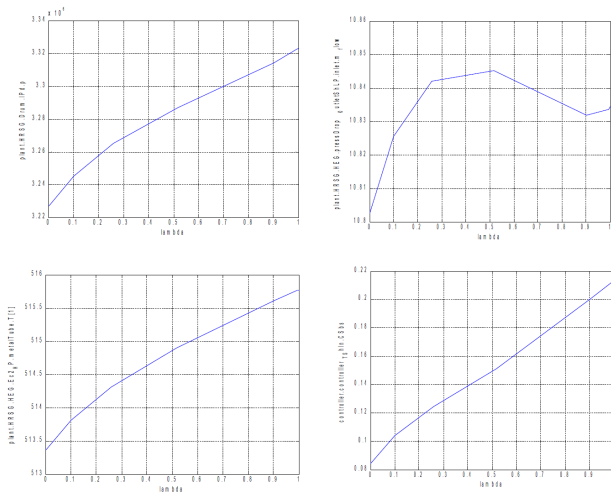


Figure 2: Homotopy paths for 100% load initialization

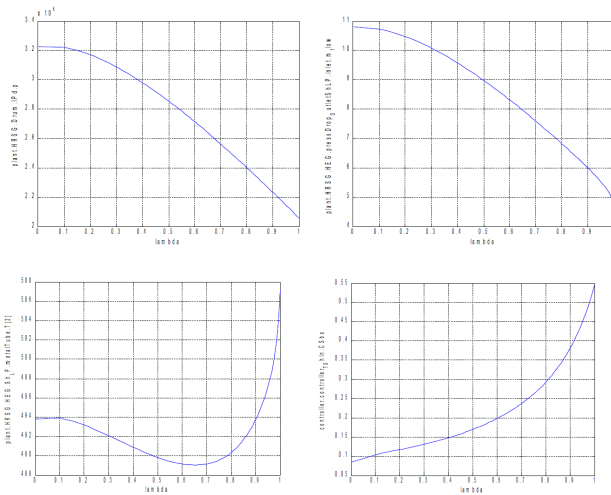


Figure 3: Homotopy paths for 60% load initialization

3 shows the plots of the same iteration variables considered in the previous case: it is apparent how the change in the values is now substantial, since it involves a large change in the operating point, but the transformation is nevertheless smooth and without singularities.

Case 3, which includes two HRSG models in parallel, has 673 iteration variables. The transformation took 7 steps and 170 seconds of CPU time to be performed. The plots of the iteration variables are similar to those shown in Figure 2 as expected, since also in this case the system is initialized at 100% load.

As a final consideration, note that the experimental code uses a brute-force numerical approach to compute the Jacobian, which can be computed in a much more efficient way by exploiting its sparsity pattern. Furthermore, not much time has been devoted to the optimal setting of the continuation solver.

A production-quality implementation is therefore expected to be substantially faster to perform the transformation shown above.

5 Conclusions and outlook

A strategy for robust and reliable steady-state initialization of large thermo-hydraulic system has been presented in this paper. The basic idea is to simplify a few selected equations in order to form a simplified initialization problem that is easily solved, without need of setting accurate start values for the iteration variables; subsequently, smoothly transform this problem into the actual problem of interest, getting its initialization by continuity.

The proposed strategy has been demonstrated by means of a test implementation of the `homotopy()` operator, applied to large models of combined-cycle power plants with up to 671 iteration variables. All the examined test cases were solved successfully and the homotopy paths of all the iteration variables did not show singular behaviour of any sort, thus confirming the validity of the selection criteria for the simplified model.

By adding a few more parameters to the model, indicating nominal values (without need of particular accuracy), the proposed method completely eliminated the need by the end user of setting start values on the particular problem at hand, in order to ensure convergence. The authors thus argue that they have demonstrated a truly modular and object-oriented approach to reliable steady-state initialization for large thermo-hydraulic networks.

The availability of built-in, fast and numerically well-behaved homotopy methods in Modelica tools would make this approach a lot more user-friendly than using the prototype implementation employed for this study, which was only meant to demonstrate the soundness of the proposed approach from the point of view of the mathematical modelling involved.

6 Acknowledgement

The financial support of EDF under contract 5900058671 (Development of an efficient method for power plant modelling) is gratefully acknowledged by the first and last authors.

References

- [1] Sielemann M., Casella F., Otter M., Clauss C., Eborn J., Mattsson S.E., Olsson H., Robust Initialization of Differential-Algebraic Equations Using Homotopy. Submitted to Modelica Conference 2011, Dresden, Germany, 20–22 March 2011.
- [2] Heroux M. A., Bartlett R. A., Howle V. E., Hoekstra R. J., Hu J. J., Kolda T. G., Lehoucq R. B., Long K. R., Pawlowski R. P., Phipps E. T., Salinger A. G., Thornquist H. K., Tuminaro R. S., Willenbring J. M., Williams A., Stanley K. S. An overview of the Trilinos project, *ACM Transactions on Mathematical Software*, 31, 397–423, 2005.
- [3] Chow S. N., Mallet-Paret J., Yorke J. A. (1978): Finding Zeroes of Maps: Homotopy Methods That are Constructive With Probability One. *Mathematics of Computation* 32, pp. 887-899, 1978.
- [4] Casella F., Leva A., Modelica Open Library For Power Plant Simulation: Design And Experimental Validation. *Proceedings 3rd International Modelica Conference*, Linköping, Sweden, Nov. 3-4, 2003, pp. 41-50.

Improving Newton's method for Initialization of Modelica models

Johan Ylikiiskilä[†]
 Johan Åkesson^{*†}, Claus Führer^{**}

* Departement of Automatic Control, Lund University

** Departement of Numerical Analysis, Lund University

<p>[†] Modelon AB Ideon Science Park SE-22370, Lund, Sweden E-mail: info@modelon.se</p>	<p>*/** Lund University Sölvegatan 18 SE-22100, Lund, Sweden E-mail: claus@maths.lth.se</p>
---	--

Abstract

Initializing a model written in Modelica translates to finding consistent initial values to the underlying DAE. Adding initial equations and conditions creates a system of non-linear equations that can be solved for the initial configuration. This paper reports an implementation of Newton's method to solve the non-linear initialization system. This implementation also uses a regularization method to deal with singular Jacobians as well as sparse solvers to exploit the sparsity structure of the Jacobian. The implementation is based on the open-source projects JModelica.org and Assimulo, KINSOL from the SUNDIALS suite and SuperLU.

Keywords: initialization; Newton's method; regularization; JModelica.org; Assimulo; KINSOL; SuperLU

1 Introduction

The initialization of a Modelica model is equivalent to finding consistent initial values to the underlying DAE:

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

Here $\mathbf{x} \in \mathbb{R}^{n_x}$ are the states and $\dot{\mathbf{x}} \in \mathbb{R}^{n_x}$ their time derivatives. $\mathbf{w} \in \mathbb{R}^{n_w}$ are the algebraic variables and t is the time.

In JModelica.org, initialization is performed by creating a system of, often non-linear, equations

called the initialization system:

$$\mathbf{F}_0(\dot{\mathbf{x}}, \mathbf{x}, \mathbf{w}, t) = 0 \quad (2)$$

The system \mathbf{F}_0 consists of the equations describing the derivatives and algebraic variables in (1), and in addition, \mathbf{F}_0 also contains information such as initial equations and fixed start values. How \mathbf{F}_0 is formed is explained in Section 2. The non-linear system of equations (2) can be solved in a multitude of ways. This paper focuses on one of the most common, Newton's method.

To simplify notation the three vectors solved for, $\dot{\mathbf{x}}$, \mathbf{x} and \mathbf{w} , are grouped together by the notation $\mathbf{u} = [\dot{\mathbf{x}}; \mathbf{x}; \mathbf{w}]$ with $\mathbf{u}_0^{(k)}$ being the values of $\dot{\mathbf{x}}$, \mathbf{x} and \mathbf{w} at time $t = 0$ and iteration k .

Being initialized by an initial guess \mathbf{u}_0 , Newton's method is basically an iteration over the following three steps:

1. Calculate a direction $\mathbf{u}_0^{(0)}$ by solving

$$\mathbf{J}(\mathbf{u}_0^{(k)}) \Delta \mathbf{u} = -\mathbf{F}_0(\mathbf{u}_0^{(k)}) \quad (3)$$

where $\mathbf{J}(\mathbf{u}_0^{(k)})$ and $\mathbf{F}_0(\mathbf{u}_0^{(k)})$ are the Jacobian and the residual calculated at the current iterate k .

2. Update $\mathbf{u}_0^{(k+1)}$:

$$\mathbf{u}_0^{(k+1)} = \mathbf{u}_0^{(k)} + \mu \Delta \mathbf{u} \quad (4)$$

where $0 < \mu \leq 1$ is a parameter that is used to increase the convergence radius, for example using linesearch [5]. If $\mu = 1$ then the method is Newton's classical method.

3. Check for convergence, and if the stopping criteria are fulfilled, return the result.

This paper will, as the title suggests, improve Newton's method to suit it better to initializing models written in Modelica. The improvement is focused on the first of the three steps constituting Newton's method, the solving of equation (3). Two issues are treated:

- An initial guess sometimes results in a singular Jacobian, so that the linear equation system no longer has a unique solution or a solution at all. In these cases a special regularization procedure has to be applied before the linear system can be numerically solved by e.g. LU factorization. This will be discussed in this paper.
- For large Modelica models the matrix $\mathbf{J}(\mathbf{u}_0^{(k)})$ is sparse, a justification for this will be given later. In this case it is interesting to look at representing the matrix $\mathbf{J}(\mathbf{u}_0^{(k)})$ in a sparse format and using a sparse linear solver such as SuperLU [12]. This paper will discuss whether, or when, such an implementation is advantageous or not.

2 JModelica.org and initialization

The initialization problem is generated in JModelica.org upon compilation. The system to be solved at initialization is (1) with additional initial equations supplied by the user. The functions associated with the initialization system, such as \mathbf{F}_0 and its Jacobian, are supplied by the JMI interface [14].

JModelica.org sets up the DAE system in its index-1 form, a form in which differential variables, x and algebraic variables w can be clearly distinguished. The system contains equations describing all derivatives and algebraic variables. It will then have $n_x + n_w$ equations resulting in an underdetermined system. Thus n_x additional equations are needed [17].

The assumption of (1) being of index 1 can be justified by saying that if a DAE of higher index

is encountered, an algorithm such as the one described in [18] is applied to reduce the problem back to a DAE of index 1.

The additional n_x equations can be supplied by the user as fixed start values and initial equations. Adding this information to the System (1), the initialization System (2) is obtained. This is done by adding all equations defined as initial equations as well as an equation of the kind (5) for each variable x_i with a modifier such as (`start = x_0, fixed = True`).

$$0 = x_i - x_0 \quad (5)$$

If the user has supplied enough additional data, the System (2) can be generated. If, however the user supplies too much information the system becomes overdetermined and the compiler will give an error message. If, on the other hand, not enough information is supplied the system becomes underdetermined. In this case the compiler will try to add information, such as setting some variables to `fixed = true`, making the system well defined. This is accomplished by applying an algorithm to compute a maximal matching between variables and equations. For this purpose, an implementation of the Hopcroft Karp matching algorithm, [11], is employed. If unmatched variables are detected, the corresponding `fixed` attributes are set to `true`, and thereby balancing the system.

3 Implementation

3.1 Overview

The implementation of the algorithm reported spans multiple packages, written in two different programming languages: Python and C. A third language, Cython [4], is used so packages written in the two different languages can communicate with each other. The algorithm basically consists of four packages, JModelica.org, Assimulo, KINSOL and an external linear solver (cf. Section 3.2) implementing a regularization method and using SuperLU.

The JModelica.org project is the biggest part and consists of code written in multiple languages, the part of JModelica.org used in this thesis is however entirely coded in Python

Assimulo is a package written in Python using

Cython to interface functionality from the SUNDIALS suite, for example KINSOL [4, 1].

Finally KINSOL and SuperLU are two packages entirely written in C. An overview of how these packages interact is presented in Figure 1.

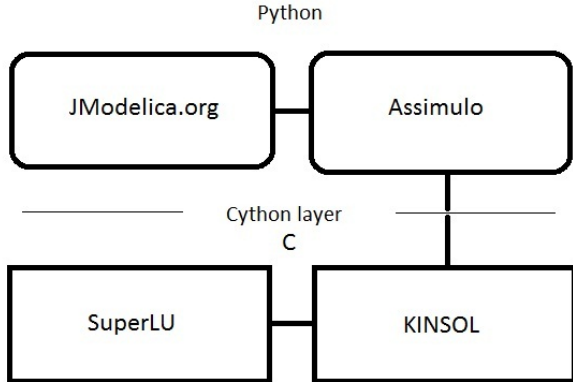


Figure 1: Overview of the packages involved in this paper and how they interact.

Model data, such as evaluation of \mathbf{F}_0 and its Jacobian, are obtained by the JMI interface in the JModelica.org part of Figure 1. The data is passed to Assimulo who calls KINSOL using Cython. In KINSOL the data is used to create the system (3) which is solved by the external linear solver, called SuperLU in Figure 1.

3.2 KINSOL

The non-linear solver implemented in the initialization algorithm reported is based on KINSOL from the SUNDIALS suite [5]. Although this report focuses on regularization and sparse solvers, some necessary theory on KINSOL has to be reviewed to allow discussion of the implementation of regularization and SuperLU.

KINSOL is a solver of systems of non-linear equations which implements a modified Newton method where the Jacobian is only evaluated when the solution progresses to slow or a certain number of iterations is exceeded [5]. This is to speed up the solution of the nonlinear system since Jacobian evaluations are expensive. The Jacobian can either be calculated by finite differences or have to be supplied as a function by the user.

A regularization method and SuperLU are implemented in KINSOL as an external linear solver. An external linear solver is called by KINSOL to solve (3) and must implement a set of functions.

The two functions that are of interest in the implementation discussed here are the *setup* and *solve* functions.

- The *setup* function is called whenever KINSOL needs to (re)evaluate the Jacobian. LU factorization is preferably performed in this function.
- The *solve* function uses the data from the last call to *setup* to solve the linear system.

4 Regularization

When in a step, say in the k^{th} step and the Jacobian is singular, the linear system (3) has no solution or its solution is not unique. Thus a different algorithm for determining the Newton increment Δu has to be used.

We require that Δu is a descent direction and a solution of the following regularized normal equations

$$\begin{aligned} \left(\mathbf{J} \left(\mathbf{u}_0^{(k)} \right)^T \mathbf{J} \left(\mathbf{u}_0^{(k)} \right) + \lambda_k \mathbf{I} \right) \Delta \mathbf{u} (h) \\ = -\mathbf{J} \left(\mathbf{u}_0^{(k)} \right)^T \mathbf{F}_0 \left(\mathbf{u}_0^{(k)} \right) \end{aligned} \quad (6)$$

with $\lambda_k > 0$.

Here, the matrix $\left(\mathbf{J} \left(\mathbf{u}_0^{(k)} \right)^T \mathbf{J} \left(\mathbf{u}_0^{(k)} \right) + \lambda_k \mathbf{I} \right)$ is positive definite with eigenvalues in $\left[\lambda_k, \lambda_k + \left\| \mathbf{J} \left(\mathbf{u}_0^{(k)} \right) \right\|_2 \right] \subset \mathbb{R}$, [19].

We select λ_k in accordance to a strategy used, when implementing the Levenberg-Marquardt method (LM) for solving an overdetermined non-linear equations systems [7, Ch. 10], by setting

$$\lambda_k := \min \left(1, \left\| \mathbf{J} \left(\mathbf{u}_k \right)^T \mathbf{F}_0 \left(\mathbf{u}_k \right) \right\| \right) \quad (7)$$

Note, in the DAE initialization process this regularization technique is required in a single, exceptional step only, while the overall process remains classical Newton iteration, based on solving regular linear systems.

4.1 Implementation

As mentioned in Section 3, regularization is implemented in an external linear solver to KINSOL. This is done so that when the LU factorization in

the *setup* function fails due to the Jacobian being singular, the regularization algorithm is called.

When the regularization is called, the regularization parameter λ_k is given by (7). Secondly, the regularized matrix $\left(\mathbf{J}(\mathbf{u}_0^{(k)})^T \mathbf{J}(\mathbf{u}_0^{(k)}) + \lambda_k \mathbf{I}\right)$ is calculated and stored as the problem Jacobian. A flag is also set telling the linear solver that the problem is currently regularized.

When the *solve* function is called it will continue as usual if the regularization flag is not set. If the flag is set however, a new right hand side corresponding to the right hand side of (6) is calculated and solved for instead of the ordinary $-\mathbf{F}_0(\mathbf{u}_0^{(k)})$.

With the regularization parameter calculated as described, there is still a problem of the Jacobian being singular at the solution. The strategy chosen is only valid for overdetermined systems if the Jacobian is regular at the solution [7, Ch. 10]. To see what effects this presents on the DAE initialization, the algorithm has been tested on the following problem (8).

$$\begin{aligned} 0 &= x^2 \\ 0 &= y^2 \end{aligned} \tag{8}$$

Problem (8) has the solution $x = y = 0$ where its Jacobian (9) is singular.

$$\begin{bmatrix} 2x & 0 \\ 0 & 2y \end{bmatrix} \tag{9}$$

The algorithm converges, albeit slowly (29 iteration when starting at $x = y = 1.0$), to the solution $x = y = 0.0018$. The stopping criteria attained in this case is the norm of the residual being smaller than a given tolerance ϵ , in this case set to $6 \cdot 10^{-6}$.

Hence the problem of a singular Jacobian at the solution slows down the algorithm but it does not cause it to crash, as long as the tolerance is not set too small. There are methods discussed in [10] handling this problem which may be included in a later implementation.

4.2 A simple example

To test if regularization indeed works, a simply constructed system with poorly chosen initial values is initialized. The example contains two states x , and y as well as an algebraic variable w and is written as follows:

```
model SingularTest
```

```
Real x ;
Real y (start = 1, fixed = true);
Real w (start = 2, fixed = true);
equation
  der(x) = x^2 - y;
  der(y) = x^2 + z^2;
  0 = w - x^2 -y;
end SingularTest;
```

In the initialization problem, the consistent values of the two states y and x , their derivatives and the algebraic variable w are solved for. The sought DAE equations are the three equations in the `equation` block. Added to these are the two equations corresponding to fixed start values. Five variables and five equations make up the well defined initialization system (10).

$$\begin{aligned} 0 &= x^2 - y - \dot{x} \\ 0 &= x^2 - z^2 - \dot{y} \\ 0 &= w - x^2 y - y \\ 0 &= y - 1 \\ 0 &= w - 2 \end{aligned} \tag{10}$$

At the initial guess given in the Modelica code (the variable x is without a start guess and is given the default guess zero), the system (10) has the Jacobian (11):

$$\begin{pmatrix} 1 & 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 & 4 \\ 0 & 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{11}$$

(11) is singular and so is $\mathbf{J}^T \mathbf{J}$, $\mathbf{J}^T \mathbf{J} + h^2 \mathbf{I}$ is however regular.

Trying to initialize this model in JModelica.org yields the regularization algorithm to be called followed by the Jacobian becoming regular and Newton's method proceeding as usual. Hence the regularization implemented succeeds in handling the singular Jacobian.

5 Sparse solvers

Another aspect taken into account when solving (3) is the structural properties if the Jacobian. When solving large systems, the solution of the system (3) can become very costly and slow due to the size of the Jacobian. But although the Jacobian is big in size it is not necessarily dense. A matrix, and the corresponding linear system, is

said to be sparse if there are many zero entries and only a few entries different from zero and this is something that can be exploited.

Another approach to exploiting model structure commonly employed in Modelica tools is based on the Block Lower Triangular (BLT) transformation, in which the system of equations is decomposed into a sequence of smaller equation systems, see e.g. [8, 9].

5.1 Sparse Jacobians in Modelica models

In the case of a Jacobian, the number of non-zero entries in row i corresponds to the number of variables upon which the function i in the system (2) is dependent. Since each function in a Modelica model of size n normally depends on about five to ten variables, the number of non-zero entries will grow linearly while the size of the Jacobian will grow quadratically for each added function. In addition, many equations in (2) come from initial values set by a `fixed = true` which only depend on one variable. The hypothesis stated here is that the Jacobian will get more sparse as the Modelica model itself gets bigger.

5.2 SuperLU

Since the Jacobians are sparse, it is interesting to look at sparse solvers for solving (3). The solver investigated in this paper is SuperLU, a fast LU-factorization algorithm optimizing memory usage [12]. The SuperLU solver will also be coupled with regularization to be able to handle singular Jacobians.

SuperLU is implemented, similar to the regularization method, as an external linear solver. Since JModelica.org has support for sparse Jacobians through the JMI interface [14], the implementation is similar to the dense case. The function calculating the sparse Jacobian is wrapped in Cython [4] and passed to KINSOL instead of the dense Jacobian. In this case the Jacobian given by the JMI interface is given in coordinate or triplet format, each non zero element is stored as the value along with the row and column number. The format required by SuperLU is *Compressed Column format* or *Harwell-Boeing format* where the columns are stored in one array, their row numbers in one array and the index of when the column changes in a third array [13]. This requires the Jacobian to

be reformatted before passing to SuperLU, which is performed by `scipy`. The computational effort for this transformation grows linearly with the size of the problem [6].

In the external linear solver, the methods used for LU factorization are called in the `setup` and the solving routines are called in the `solve` function. Regularization is also implemented in the same fashion as in the dense solver but with sparse matrices.

6 Results

6.1 Regularization

As mentioned briefly in the end of Section 4.2, the regularization algorithm succeeds with the constructed example presented there. A model of a distillation column, `jmodelica.examples.distillation`, from the JModelica.org distribution, is a model with a singular Jacobian at the initial guess supplied in the Modelica file. When solving the initialization problem with KINSOL coupled with an ordinary linear solver, the solver fails, stating that the Jacobian could not be LU-factorized. When a regularized linear solver, like the ones described in Sections 4.1 and 5.2, are used however, one regularized step is taken and KINSOL then converges to the solution without having to perform another regularization step.

6.2 SuperLU

To test the efficiency of the initialization algorithm with SuperLU, several Modelica models have been initialized with the sparse and the dense initialization algorithm. The initialization has been timed multiple times and a mean value is calculated. The mean values and medians of the times are later compared to decide which algorithm is faster. The tests have been performed on a Intel Core 2 Duo T5870 processor under 32 bit Windows 7 Professional.

To test if the initialization algorithm is faster using SuperLU instead of a dense linear solver two series of non-linear systems have been compared. From [2] the problem series *Broyden* and *Moraeux* are problems concerning constrained optimization but can be seen as a non linear root finding problem. A simple script `Atom.py` is implemented to translate the models, supplied in the

AMPL format `.mod`, to Modelica for later treatment by JModelica.org. These problem series offer similar systems of different size. In Figures 2 and 3 the speedups of both the Broyden and Moreaux problems are plotted against the number of variables of the systems. In Figure 2 the speedup of the total time is plotted while Figure 3 plots the speedup of the total time except the time spent evaluating system functions and Jacobians. Here speedup means the time measured with dense solver divided by time measured with sparse solver, i.e. how many times faster the sparse solver is than the dense solver. It should be noticed that computation of Jacobians in JModelica.org used for the benchmarks is slow, due to limitations in the CppAD package, [3], with regards to sparse Jacobians. Therefore, we focus on comparison of the time spent in KINSOL in the cases of sparse versus dense linear solvers.

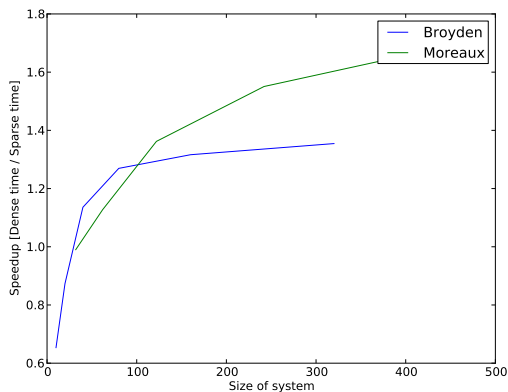


Figure 2: Speedup of the total times for the Broyden and Moreaux problems.

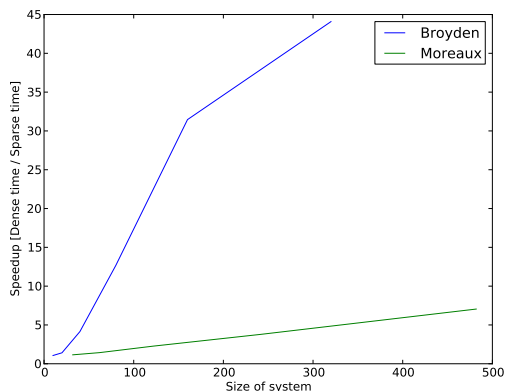


Figure 3: Speedup of the Broyden and Moreaux problems not counting time in fevals and jevals.

In Tables 1 and 2 the data from the test runs on the Broyden and Moreaux problems are presented. The data consists of the time spent in total is presented (tot) along the time spent in KINSOL and linear solver without evaluations of Jacobians and system functions (KIN), the time spent on evaluating the residual and Jacobian (Evals) and the number of non-linear iterations required (iters). The data is scaled by the total time of the dense solver to simplify comparison.

Table 1: Times measured for the Broyden problems.

	Broyden	10	40	80	320
	size	10	40	80	320
tot	Dense	1.0	1.0	1.0	1.0
	Sparse	1.530	0.880	0.787	0.738
KIN	Dense	0.452	0.249	0.244	0.243
	Sparse	0.425	0.060	0.019	0.006
Evals	Dense	0.548	0.751	0.756	0.757
	Sparse	1.105	0.820	0.768	0.732
iters	Dense	17	60	112	137
	Sparse	17	60	112	137

Table 2: Times measured for the Moreaux problems.

	Moreaux	10	40	80	160
	size	32	122	242	482
tot	Dense	1.0	1.0	1.0	1.0
	Sparse	1.010	0.734	0.645	0.584
KIN	Dense	0.666	0.495	0.457	0.435
	Sparse	0.580	0.218	0.121	0.062
Evals	Dense	0.334	0.505	0.543	0.565
	Sparse	0.430	0.516	0.524	0.522
iters	Dense	38	112	123	137
	Sparse	38	112	123	137

The models tested in Tables 1 and 2 are not models originally written in Modelica but rather optimization benchmarks. To test how the initialization algorithm using a sparse solver behaves when used on 'real' Modelica models, the same test performed in Tables 1 and 2 are performed on some models with different sizes in Table 3.

- CSTR: an example from the JModelica.org package describing two continuously stirred tank reactors in series.
- DIST: an example from the JModelica.org package already mentioned in Section 6.1.

- CoCy: a Modelica model describing a combined cycle power plant initialized at full load.

Table 3: Times measured for the Modelica models.

	Model	CSTR	Dist	CoCy
	size	15	99	150
tot	Dense	1.0	1.0	1.0
	Sparse	1.112	0.599	0.635
KIN	Dense	0.645	0.552	0.426
	Sparse	0.610	0.115	0.112
Evals	Dense	0.355	0.448	0.574
	Sparse	0.502	0.484	0.523
iters	Dense	10	24	34
	Sparse	10	24	34

6.3 Sparsity of Jacobians

It is also interesting to take a look at the sparsity of the systems to put the results obtained in Section 6.2. In Table 4 the sparsity of the systems, that is the sparsity of the system Jacobian, timed in Table 1 and 2, are presented.

Table 4: Sparsity measured in the percentage of elements different from zero in the Jacobian.

Model	10	20	40	80	160	320
Broyden	54.0	31.0	16.5	8.5	4.31	2.17
Moreaux	8.98	4.73	2.43	1.23	0.62	-

In Table 5 the sparsity of the Modelica models timed in Table 3 are presented supporting the hypothesis of bigger models being more sparse..

Table 5: Sparsity measured in the percentage of elements different from zero in the Jacobian.

Model	Size	Sparisty
CSTR	15	24.0
DIST	99	2.67
CoCy	150	1.75

7 Conclusions

The regularization method is handling singular Jacobians at initialization. So far, no models, supported by JModelica.org, have caused the initialization algorithm based on regularization to stop due to a singular Jacobian. A problem with a singular Jacobian at the solution is however solved

slower, as shown in the end of Section 4.1. Pan and Fan [10] proposes techniques to handle this problem that may be used in a later implementations.

Table 5 imply that larger Modelica models are more sparse than smaller Modelica models, thus supporting the hypothesis stated in section 5.1 of Modelica models getting more sparse as they grow in size.

Regarding sparsity, Figures 2, 3 and Tables 1, 2 and 3 imply that the problems are initialized faster with the sparse version of the initialization algorithm. Due to CppAD slowing down the evaluations of Jacobian, the times spent in KINSOL are compared instead of the total time.

When applied to the Modelica models in Table 3, the sparse version solves the bigger problems (of size $n \approx 100$ or bigger) around 4-5 times faster than the dense version. The bigger benchmarks from the Broyden and Moreaux series show an even bigger speedup, Broyden320 is for example solved 40 times faster. For smaller model, like the model of the two stirred tank reactors, the organizational effort of SuperLU and the model being less sparse, outweighs the advantages and the two methods are equal.

In the benchmarks presented here, the time for evaluating Jacobians outweighs the time spent in KINSOL, especially if SuperLU is employed. This is due to the fact that the package used for generation of Jacobians has weak support for computation of sparse derivatives. This deficiency will be addressed in future versions of JModelica.org.

In conclusion, the sparse version of the initialization algorithm is advantageous when applied to bigger models. For smaller models however, the two version performs equally. However, the slow evaluation of sparse Jacobians make the dense solver a better choice for smaller models.

References

- [1] The assimulo homepage. <http://www.jmodelica.org/page/199>.
- [2] The coconut benchmark: Library 3 constraint satisfaction test problems. http://www.mat.univie.ac.at/~neum/glopt/coconut/Benchmark/Library3_new_v1.html.
- [3] The cppad webpage. <http://www.coin-or.org/CppAD/>.

- [4] Stefan Behnel, Robert Bradshaw, Dag Sverre Seljebotn, and other contributors. Cython c-extensions for python - homepage. <http://www.cython.org/>.
- [5] Aaron M. Collier, Alan C. Hindmarsh-hand Radu Serban, and Carol S. Woodward. *User Documentation for kinsol v2.6.0*. Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, May 2009.
- [6] The SciPy community. *SciPy Reference Guide*, 0.9.0.dev6665 edition, October 2010.
- [7] J.E. Dennis and R.B. Schnabel. *Numerical methods for unconstrained optimization and nonlinear equations*. Prentice Hall, 1983.
- [8] Hilding Elmqvist. *A Structured Model Language for Large Continuous Systems*. PhD thesis, Department of Automatic Control, Lund University, Sweden, may 1978. TFRT-1015.
- [9] Jan Eriksson. A note on the decomposition of systems of sparse non-linear equations. *BIT Numerical Mathematics*, 16(4):462–465, 1976. 10.1007/BF01932730.
- [10] Jinyan Fan and Jianyu Pan. A note on the levenberg-marquardt parameter. *Applied Mathematics and Computation*, 207:351–359, 2009.
- [11] John E. Hopcroft and Richard M. Karp. An $2^{2/5}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973.
- [12] X. S. L. W. Demmel James W. Demmel, John R. Gilbert, Stanley C. Eisenstat, John R. Gilbert, Xiaoye S. Li, and Joseph W. H. Liu. A supernodal approach to sparse partial pivoting. *SIAM J. MATRIX ANAL. APPL.*, 20(3):720–755, 1999.
- [13] Xiaoye S. Li James W. Demmel, John R. Gilbert. *SuperLU Users Guide*.
- [14] 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, nov 2010.
- [15] K. Levenberg. A method for the solution of certain nonlinear problems in least squares. *Quart. Appl. Math.*, 2:164–166, 1944.
- [16] D.W. Marquardt. An algorithm for least-squares estimation of nonlinear inequalities. *SIAM J. Appl. Math.*, 11:431–441, 1963.
- [17] S.E Mattson, H. Elmqvist, M. Otter, and H. Olsson. Initialization of hybrid differential-algebraic equations in modelica 2.0. In *Second International Modelica Conferencem, Proceedings*, pages 9–15. The Modelica Association, March 2002.
- [18] Sven Erik Mattsson and Gustaf Söderlind. Index reduction in differential-algebraic equations using dummy derivatives. *SIAM J. Sci. Comput*, 14(3):677–692, May 1993.
- [19] Arnold Neumaier. Solving ill-conditioned and singular linear systems: A tutorial on regularization. *SIAM Review*, 40(3):636–666, September 1998.
- [20] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, 2006.

Acknowledgements

The authors would like to acknowledge the kind assistance from Francesco Casella in providing the Combined Cycle benchmark model used in the paper. This work was partially funded by the Swedish funding agency Vinnova under the grant program "Forska and Väx".

The Functional Mockup Interface for Tool independent Exchange of Simulation Models

T. Blochwitz¹, M. Otter², M. Arnold³, C. Bausch⁴, C. Clauß⁵, H. Elmqvist⁹, A. Junghanns⁶,
J. Mauss⁶, M. Monteiro⁴, T. Neidhold¹, D. Neumerkel⁷, H. Olsson⁹, J.-V. Peetz⁸, S. Wolf⁵

Germany: ¹ITI GmbH, Dresden; ²DLR Oberpfaffenhofen; ³University of Halle,
⁴Atego Systems GmbH, Wolfsburg; ⁵Fraunhofer IIS EAS, Dresden; ⁶QTronic, Berlin;
⁷Daimler AG, Stuttgart; ⁸Fraunhofer SCAI, St. Augustin;

Sweden: ⁹Dassault Systèmes, Lund.

Abstract

The Functional Mockup Interface (FMI) is a tool independent standard for the exchange of dynamic models and for co-simulation. The development of FMI was initiated and organized by Daimler AG within the ITEA2 project MODELISAR. The primary goal is to support the exchange of simulation models between suppliers and OEMs even if a large variety of different tools are used. The FMI was developed in a close collaboration between simulation tool vendors and research institutes. In this article an overview about FMI is given and technical details about the solution are discussed.

Keywords: Simulation; Co-Simulation, Model Exchange; MODELISAR; Functional Mockup Interface (FMI); Functional Mockup Unit (FMU);

1 Introduction

One of the objectives of the development and usage of tool independent modeling languages (e.g. Modelica[®][1]¹, VHDL-AMS [10]) is to ease the model exchange between simulation tools. However, modeling languages require a huge effort to support them in a tool. It is therefore common to provide also low level interfaces, to exchange models in a less powerful, but much simpler way. Another aspect of model exchange is the protection of product know-how which could be recovered from their physical models.

Several tools offer proprietary model interfaces, such as:

- Matlab/Simulink^{®2}: S-Functions [3]
- MSC.ADAMS³: user-written subroutines [4]
- Silver: Silver-Module API [5]
- SIMPACK: user routines [6]
- SimulationX^{®4}: External Model Interface [7]

Currently, no tool independent standard for model exchange (via source or binary code in a programming language) is available. The same holds for the situation in the field of co-simulation.

Vendors of Modelica tools (AMESim, Dymola, SimulationX) and non Modelica tools (SIMPACK, Silver, Exite), as well as research institutes worked closely together and recently defined the Functional Mockup Interface⁵. This interface covers the aspects of model exchange [8] and of co-simulation [9]. This development was initiated and organized by Daimler AG with the goal to improve the exchange of simulation models between suppliers and OEMs. Within MODELISAR, Daimler has set up 14 automotive use cases for the evaluation and improvement of FMI. In this article, the technical details behind FMI are discussed.

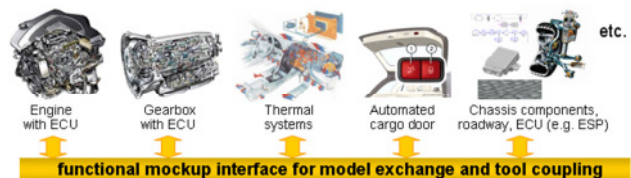


Figure 1: Improving model-based design between OEM and supplier with FMI.

¹ Modelica[®] is a registered trademark of the Modelica Association.

² Matlab[®]/Simulink[®] are regist. trademarks of The MathWorks Inc.

³ MSC[®] is a registered trademark and MSC.ADAMS is a trademark of MSC Software Corporation or its subsidiaries.

⁴ SimulationX[®] is a registered trademark of ITI GmbH.

⁵ <http://www.functional-mockup-interface.org>

2 The Functional Mock-Up Interface

2.1 Main Design Ideas

The FMI 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 form of an input/output block that can be utilized by other modeling and simulation environments. Models are described by differential, algebraic and discrete equations with time-, state- and step-events. The models to be treated can be large for usage in offline simulation; and it is also possible to use models for online simulation and in embedded control systems on micro-processors.
2. *FMI for Co-Simulation:*
The intention is to couple two or more simulation tools in a co-simulation environment. The data exchange between subsystems is restricted to discrete communication points. In the time between two communication points, the subsystems 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 (slaves). 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.

Both approaches share a bulk of common parts that

are sketched in the next subsections.

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:

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. For FMI-for-Co-Simulation, all information about the “slaves”, which is relevant for the communication in the co-simulation environment is provided in a slave specific XML-file. In particular, this includes a set of capability flags to characterize the ability of the slave to support advanced master algorithms, e.g. the usage of variable communication step sizes, higher order signal extrapolation, or others.
2. For the FMI-for-Model-Exchange case, all needed model equations are provided with a small set of easy to use C-functions. 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.
For the FMI-for-Co-Simulation case, also a small set of easy to use C-functions are provided in source and/or binary form to initiate a communication with a simulation tool, to compute a communication time step, and to perform the da-

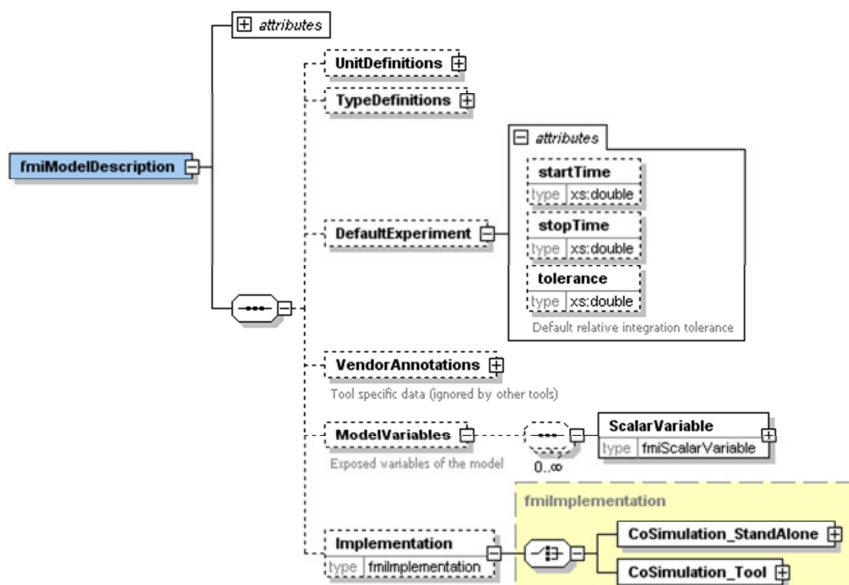


Figure 2: Top level part of the FMI XML schema

ta exchange at the communication points.

- 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. As usual, the XML-file is defined by an XML-schema file called “fmiModelDescription.xsd”. Most information is identical for the two FMI cases.

In Figure 2, the top-level part of the schema definition is shown. All parts are the same for the two FMI-cases, with exception of the element “Implementation”. If present, the import tool should interpret the model description as applying to co-simulation. As a consequence, the import tool must select the C-functions for co-simulation, otherwise for model exchange. An important part of the “Implementation” is the definition of capability flags to define the capabilities that the co-simulation slave supports:



Figure 3: Capability flags of FMI for Co-Simulation.

These flags are interpreted by the master to select a co-simulation algorithm which is supported by all connected slaves.

2.4 C-Interface

The executive part of FMI consists of two header files that define the C-types and -interfaces. The header file “fmiPlatformTypes.h” contains all definitions that depend on the target platform:

```
#define fmiPlatform "standard32"
#define fmiTrue 1
#define fmiFalse 0
#define fmiUndefinedValueReference
        (fmiValueReference) (-1)

typedef void* fmiComponent;
typedef unsigned int fmiValueReference;
typedef double fmiReal ;
typedef int fmiInteger;
typedef char fmiBoolean;
typedef const char* fmiString ;
```

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.

In this first version of FMI, the minimum amount of different data types is defined. This is not sufficient for embedded systems and will be improved in one of the follow-up versions of FMI.

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 (like fmiReal) besides alias variables that have the same handle. All structured entities, like records or arrays, are “flattened” in to a set of scalar values of type fmiReal, fmiInteger etc. A fmiValueReference references one such scalar. The coding of fmiValueReference is a “secret” of 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, and Boolean. One argument of these functions is an array of fmiValueReference, which defines which variable is accessed. The mapping between the FMU variables and the fmiValueReferences is stored in the model description XML file.

For simplicity, in this first version of FMI 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 an element of an array of records “pipe” of vector T (“.” separates hierarchical levels and “[...]” defines array elements).

Header-file “fmiFunctions.h” contains the prototypes for functions that can be called from simulation environments.

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 unless function pointers must be used. For simplicity and since the function pointer approach is not desirable on embedded systems, the first variant is utilized by FMI: Macros are provided in “fmiFunctions.h” to build the actual function names. A typical usage in an FMI model or co-simulation slave is:

```
#define MODEL_IDENTIFIER MyFMU
#include "fmiFunctions.h"
< implementation of the FMI functions >
```

For example, a function that is defined as

“fmiGetDerivatives”

is changed by the macros to the actual function name

“MyFMU_fmiGetDerivatives”,

i.e., the function name is prefixed with the model or slave name and an “_”. The “MODEL_IDENTIFIER” is defined in the XML-file of the FMU. A simulation environment can therefore construct the relevant function names after inspection of the XML-file. This can be used by (a) generating code for the actual function call or (b) by dynamically loading a dynamic link library and explicitly importing the function symbols by providing the “real” function names as strings.

3 FMI for Model Exchange

3.1 Mathematical Description

The goal of the Model Exchange interface is to numerically solve a system of differential, algebraic and discrete equations. In this version of the interface, ordinary differential equations in state space form with events are handled (abbreviated as “hybrid ODE”).

This type of system is described as a piecewise continuous system. Discontinuities can occur at time instants t_0, t_1, \dots, t_n , where $t_i < t_{i+1}$. These time instants are called “events”. Events can be known before hand (= time event), or are defined implicitly (= state and step events).

The “state” of a hybrid ODE is represented by a continuous state $\mathbf{x}(t)$ and by a time-discrete state $\mathbf{m}(t)$ that have the following properties:

- $\mathbf{x}(t)$ is a vector of real numbers (= time-continuous states) and is a continuous function of time inside each interval $t_i \leq t < t_{i+1}$, where $t_i = \lim_{\varepsilon \rightarrow 0} (t_i + \varepsilon)$, i.e., the right limit to t_i (note, $\mathbf{x}(t)$ is continuous between the right limit to t_i and the left limit to t_{i+1} respectively).
- $\mathbf{m}(t)$ is a set of real, integer, logical, and string variables (= time-discrete states) that are constant inside each interval $t_i \leq t < t_{i+1}$. In other words, $\mathbf{m}(t)$ changes value only at events. This means, $\mathbf{m}(t) = \mathbf{m}(t_i)$, for $t_i \leq t < t_{i+1}$.

At every event instant t_i , variables might be discontinuous and therefore have two values at this time instant, the “left” and the “right” limit. $\mathbf{x}(t_i), \mathbf{m}(t_i)$ are always defined to be the right limit at t_i , whereas $\mathbf{x}^-(t_i), \mathbf{m}^-(t_i)$ are defined to be the “left” limit at t_i , e.g.: $\mathbf{m}^-(t_i) = \mathbf{m}(t_{i-1})$. In the following figure, the two variable types are visualized:

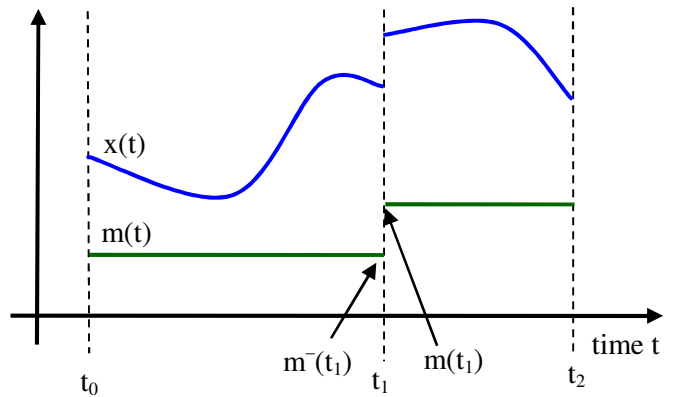


Figure 4: Piecewise-continuous states of an FMU: time-continuous (x) and time-discrete (m).

An event instant t_i is defined by one of the following conditions that gives the smallest time instant:

1. At a predefined time instant $t_i = T_{next}(t_{i-1})$ that was defined at the previous event instant t_{i-1} either by the FMU, or by the environment of the FMU due to a discontinuous change of an input signal u_j at t_i . Such an event is called time event.
2. At a time instant, where an event indicator $z_j(t)$ changes its domain from $z_j > 0$ to $z_j \leq 0$ or vice versa (see Figure 5 below). More precisely: An event $t = t_i$ occurs at the smallest time instant “min t ” with $t > t_{i-1}$ where “ $(z_j(t) > 0) \neq (z_j(t_{i-1}) >$

0)”. Such an event is called state event. All event indicators are piecewise continuous and are collected together in one vector of real numbers $z(t)$.

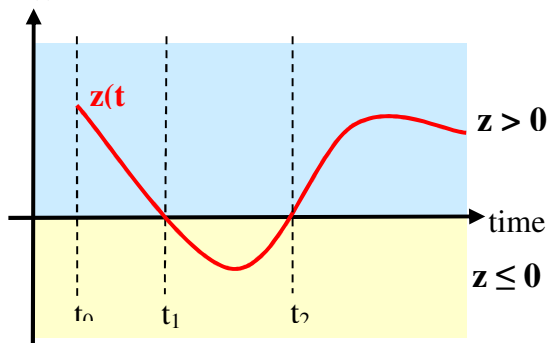


Figure 5: An event occurs when the event indicator changes its domain from $z > 0$ to $z \leq 0$ or vice versa.

3. At every completed step of an integrator, `fmiCompletedIntegratorStep` must be called. An event occurs at this time instant, if indicated by the return argument `callEventUpdate`. Such an event is called step event. Step events are, e.g., used to dynamically change the (continuous) states of a model, because the previous states are no longer suited numerically.

An event is always triggered from the environment in which the FMU is called, so it is not triggered inside the FMU. A model (FMU) may have additional variables \mathbf{p} , \mathbf{u} , \mathbf{y} , \mathbf{v} . These symbols characterize sets of real integer, logical, and string variables, respectively. The non-real variables change their values only at events. For example, this means that $u_j(t) = u_j(t_i)$, for $t_i \leq t < t_{i+1}$, if u_j is an integer, logical or string variable. If u_j is a real variable, it is either a continuous function of time inside this interval or it is constant in this interval (= time-discrete). “ \mathbf{p} ” are parameters (data that is constant during the simulation), “ \mathbf{u} ” are inputs (signals provided from the environment), “ \mathbf{y} ” are outputs (signals provided to the environment that can be used as inputs to other subsystems), and “ \mathbf{v} ” are internal variables that are not used in connections, but are only exposed by the model to inspect results. Typically, there are a few inputs \mathbf{u} and outputs \mathbf{y} (say 10), and many internal variables \mathbf{v} (say 100000).

3.2 Caching of Variables

Depending on the situation, different variables need to be computed. In order to be efficient, FMI is designed so that the interface requires only the computation of variables that are needed in the present context. For example, during the iteration of an integrator step, only the state derivatives need to be com-

puted provided the output of a model is not connected. It might be that at the same time instant other variables are needed. For example, if an integrator step is completed, the event indicator functions need to be computed as well. For efficiency it is then important that in the call to compute the event indicator functions, the state derivatives are not newly computed, if they have been computed already at the present time instant. This means, the state derivatives shall be reused from the previous call. This feature is called “caching of variables”.

Caching requires that the model evaluation can detect when the input arguments, like time or states, have changed. This is achieved by setting them explicitly with a function call since every such function call signals precisely a change of the corresponding variables. A typical call sequence to compute the derivatives

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}, \mathbf{p}, t)$$

as function of states, inputs, and parameters is therefore:

```
// Instantiate FMU
// ("M" is the MODEL_IDENTIFIER)
m = M_fmiInstantiateModel("m", ...);
...
// set parameters
M_fmiSetReal(m, id_p, np, p);

// initialize instance
M_fmiInitialize(m, ...);
...
// set time
M_fmiSetTime(m, time);
...
// set inputs
M_fmiSetReal(m, id_u, nu, u);
...
// set states
M_fmiSetContinuousStates(m, x, nx);
...
// get state derivatives
M_fmiGetDerivatives(m, der_x, nx);
```

To obtain the FMU outputs:

$$\mathbf{y} = \mathbf{f}(\mathbf{x}, \mathbf{u}, \mathbf{p}, t)$$

as function of states, inputs, and parameters, the environment would call:

```
...
// get outputs
M_fmiGetReal(m, id_y, ny, y);
...
```

The FMU decides internally which part of the model code is evaluated in the present context.

4 FMI for Co-Simulation

Co-simulation is a simulation technique for coupled time-continuous and time-discrete systems that exploits the modular structure of coupled problems in all stages of the simulation process (pre-processing, time integration, post-processing).

The data exchange between subsystems is restricted to discrete communication points. In the time between two communication points, the subsystems 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 (slaves).

Examples for co-simulation techniques are the distributed simulation of heterogeneous systems, partitioning and parallelization of large systems, multi rate integration, and hardware-in-the-loop simulations.

A simulation tool can be coupled if it is able to communicate data during simulation at certain time points (communication points, t_{c_i}), see Figure 6.

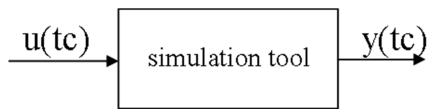


Figure 6: Coupleable simulation tool

4.1 Master Slave Principle

Instead of coupling the simulation tools directly, it is assumed that all communication is handled via a master (Figure 7).

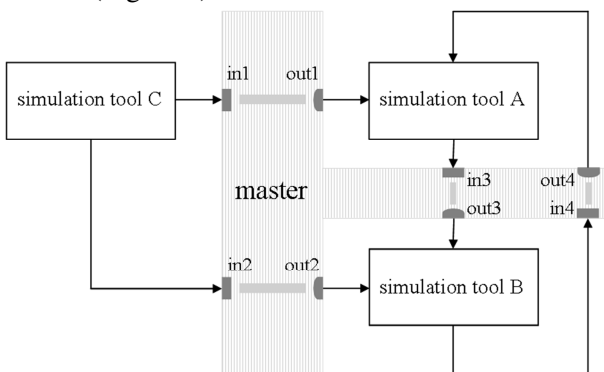


Figure 7: Three tools are controlled by one master

The master plays an essential role in controlling the coupled simulation. Besides distribution of communication data, the master analyses the connection graph, chooses a suitable simulation algorithm and

controls the simulation according to that algorithm. The slaves are the simulation tools, which are prepared to simulate their subtask. The slaves are able to communicate data, execute control commands and return status information.

4.2 Interface

The FMI for Co-Simulation defines similar functions like FMI for Model Exchange for creation, initialization, termination, and destruction of the slaves. In order to allow distributed scenarios, the co-simulation functions provide some more arguments. E.g. the function `fmiInstantiateSlave(...)` provides the string argument `mimeType` which defines the tool which is needed to compute the FMU in a tool based co-simulation scenario (see section 4.3).

For data exchange, the `fmiGet.../fmiSet...` functions of FMI for Model Exchange are used here too. In order to allow higher order extrapolation/interpolation of continuous inputs/outputs additional Get/Set functions are defined for input/output derivatives.

The computation of a communication time step is initiated by the call of `fmiDoStep(...)`. The function arguments are the current communication time instance, the communication step size, and a flag that indicates whether the previous communication step was accepted by the master and a new communication step is started. Depending on the internal state of the slave and the previous call of `fmiDoStep(...)`, the slave has to decide which action is to be done before the step is computed. E.g. if a communication time step is repeated, the last taken step is to be discarded.

Using this interface function, different co-simulation algorithms are possible:

- Constant and variable communication step sizes.
- Straightforward methods without rejecting and repetition of communication steps.
- Sophisticated methods with iterative repetition of communication steps.

4.3 Use Cases

The FMI for Co-Simulation standard can be used to support different usage scenarios. The simplest one is shown in the following figure (in order to keep it simple, only one slave is drawn).

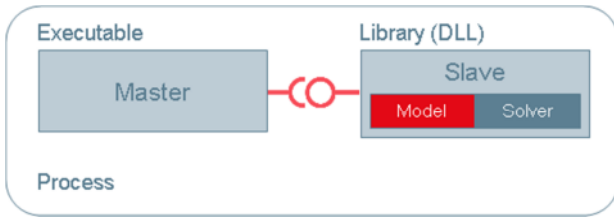


Figure 8: Simple stand alone use case.

Master and slave are running in the same process. The slave contains model and solver, no additional tool is needed.

The next use case is carried out on different processes. A complete simulation tool acts as slave. In this case, the FMI is implemented by a simulation tool dependent wrapper which communicates by a (proprietary) interface with the simulation tool.

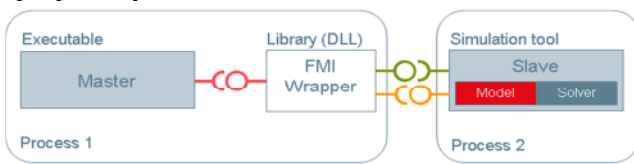


Figure 9: Slave is a simulation tool

The green and orange interfaces can be inter-process communication technologies like COM/DCOM, CORBA, Windows message based interfaces or signals and events. The co-simulation master does not notice the usage of an FMI wrapper here. It still utilizes only the FMI for Co-Simulation.

Figure 10 demonstrates a distributed co-simulation scenario.

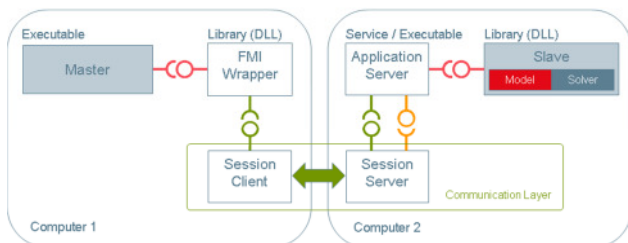


Figure 10: Distributed co-simulation scenario.

The slave (which can be a simulation tool as in Figure 9 too) and the master run on different computers. The data exchange is handled by a communication layer which is implemented by a special FMI wrapper. Neither the master nor the slave notice the technology used by the communication layer. Both utilize the FMI for Co-Simulation only.

5 Comparing Model-Exchange Approaches

As shown in section 1, there are multiple, proprietary approaches for exchanging executable models. Here

we discuss the differences with respect to the following properties:

1. Interface coverage:
 - a) Model representation as ordinary differential equation (ODE) or differential algebraic equation (DAE).
 - b) Does the API support querying Jacobian matrix information?
 - c) Is it possible to transfer structural data via the API? If information about algebraic dependencies between outputs and inputs are supplied, the importing tool is able to detect and handle algebraic loops automatically.
2. Event handling: Does the API support transporting event handling information between model and simulation environment for
 - a) Time events?
 - b) State events?
 - c) Step events?
 - d) Event iteration?
3. Step-size control: Does the API support
 - a) Rejecting of time step by the model?
 - b) Variable step sizes?
4. Efficiency: Does the API support
 - a) efficient computing (e.g. allowing value caching)?
 - b) efficient result storage (e.g. via alias mechanism and storing large numbers of internal variables)?
 - c) efficient argument handling (data copy required)?
5. Programming languages: Which programming languages are supported by the API
6. Documentation: Is documentation sufficient for
 - a) using the APIs data types?
 - b) building models with this API (export)?
 - c) using models with this API in other simulation environments (import)?
 - d) each models variables (inputs, outputs, states etc.)?
7. Miscellaneous:
 - a) Interoperability: which platforms are supported?
 - b) Does changing the (version of the) simulation environment require model re-compilation?
 - c) Compact and flexible file format, that allows the inclusion of additional data.
8. Status of API:
 - a) License?
 - b) Developed and maintained by only one tool vendor?

	Property	Simulink: S-Function	ADAMS: user routines	Silver: Silver API	SIMPACT: user routines	SimulationX: EMI	FMI for Model Exchange
1a	Representation	ODE	ODE	Co-Sim ¹	ODE/DAE	ODE	ODE ²
1b	Jacobian support	no	no	no	no	no	no ³
1c	Structural data	yes	no	no	no ⁴	yes	yes
1a	Time events	yes ⁵	no	no	yes	yes ⁵	yes ^{5,6}
1b	State events	yes	no	no	yes	yes	yes
2c	Step events	yes	no	no	yes	yes	yes
2d	Event iteration	no	no	no	yes	yes	yes
3a	Discard time step	no	no	no	no	yes	yes
3b	Variable step size	yes	yes	no	yes	yes	yes
4a	Eff. computing	no ⁷	no ⁷	no ⁷	no ⁷	no ⁷	yes
4b	Eff. Result storage	no ⁸	no	no	no ⁸	no ⁸	yes
4c	Data copy required	no	no	no	no	no	yes
5	API language	C, C++, Fortran	C, Fortran	C, Python	C, Fortran	C	C
6a	Doc API data types	yes	yes	yes	yes	yes	yes
6b	Doc model generation	yes	yes	yes	yes	yes	yes
6c	Doc model use	no	no	yes	yes	yes	yes
6d	Doc model variables	no		yes	no	no	yes
7a	Platform support	independent ⁹	Windows, Linux	Windows	Windows, UNIX	Windows	independent. ^{9,10}
7b	Recompilation	yes	yes	no	yes ¹¹	yes	no
7c	Compact file format	no	no	no	no	no	yes
8a	Legal status	proprietary	proprietary	open	proprietary	open	open
8b	Proprietary	yes	yes	yes	yes	yes	no

1. Silver supports only exchanging models including their own solvers, communicating at project-dependent, equidistant time intervals.
2. DAE support is planned for one of the next versions.
3. Planned for one of the next FMI versions.
4. Loops are detected and can be handled using additional user intervention on the model.
5. API supports time events, but definition is based on real variables which could lead to inexact results.
6. Efficient and numerically robust time event handling is planned for FMI Version 2.0
7. Caching mechanism is possible, but the component itself has to observe which data have changed since the last call.
8. API is not designed to transfer more data than inputs, outputs, and states. Internal variables are not published by the external model.
9. The component can be provided as binary for one operating system and/or as source code.
10. The model source code can be part of the FMU archive file.
11. Recompilation is only needed for major version changes.

6 Tools supporting FMI

The following tools are currently supporting the FMI version 1.0 (for an up-to-date list see www.functional-mockup-interface.org/tools.html):

- **Dymola 7.4** (FMU export and import, www.3ds.com/products/catia/portfolio/dymola),
- **JModelica.org 1.3** (FMU import, www.jmodelica.org),
- **Silver 2.0** (FMU import, www.qtronic.com/en/silver.html),
- **SimulationX 3.4** (FMU import and export, FMU co-simulation, www.simulationx.com, see [10])
- **Simulink** (FMU export by Dymola 7.4 via Real-Time Workshop, www.mathworks.com/products/simulink),
- **Fraunhofer Co-Simulation Master** (see [12])

The respective vendors plan to support FMI with their following tools:

- **AMESim** (FMU export and import, www.lmsintl.com/Imagine-AMESim-1-d-multi-domain-system-simulation),
- **EXITE, EXITE ACE** (FMU export and import, www.extessy.com),
- **OpenModelica** (FMU export and import; a prototype for FMU export is available, www.openmodelica.org),
- **SIMPACT** (FMU import, FMU co-simulation, www.simpact.com),
- **TISC** (FMU import, www.tlk-thermo.com).
- **MpCCI** (FMU import and export, www.mpcci.de)

7 Conclusions

The FMI eases the model exchange and co-simulation between different tools. It has a high potential being widely accepted in the CAE world:

- It was initiated, organized and pushed by Daimler to significantly improve the exchange of simulation models between suppliers and OEMs.
- It was defined in close collaboration of different tool vendors within the MODELISAR project.
- Industrial users were involved in the proof of concept within MODELISAR.

- FMI can already be used with several Modelica tools, Simulink, multi-body and other tools.
- In the ITEA2 project OPENPROD [10], FMI extensions are currently developed with respect to optimization applications.

Current priorities of the further development inside MODELISAR are:

- Unification and harmonization of FMI for Model Exchange and Co-Simulation
- Improved handling of time events.
- Improved support for FMUs in embedded systems.
- Efficient interface to Jacobian matrices.

8 Acknowledgements

This work is supported by the German BMBF (Förderkennzeichen: 01IS08002) and the Swedish VINNOVA (funding number: 2008-02291) within the ITEA2 MODELISAR project (http://www.itea2.org/public/project_leaflets/MODELISAR_profile_oct-08.pdf). The authors appreciate the partial funding of this work.

9 References

- [1] 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>
- [2] **VHDL-AMS: IEEE Std 1076.1-2007.** Nov. 15, 2007. VHDL-AMS web page: <http://www.vhdl.org/vhdl-ams/>
- [3] The Mathworks: **Manual: Writing S-Functions**, 2002
- [4] **Using ADAMS/Solver Subroutines.** Mechanical Dynamics, Inc., 1998.
- [5] A. Junghanns: **Virtual integration of Automotive Hard- and Software with Silver.** ITI-Symposium, 24.-25.11.2010, Dresden.
- [6] <http://www.simpact.com>
- [7] Blochwitz T., Kurzbach G., Neidhold T. **An External Model Interface for Modelica.** 6th International Modelica Conference, Bielefeld 2008. www.modelica.org/events/modelica2008/Proceedings/sessions/session5f.pdf

- [8] MODELISAR Consortium: **Functional Mock-up Interface for Model Exchange. Version 1.0**, www.functional-mockup-interface.org
- [9] MODELISAR Consortium: **Functional Mock-up Interface for Co-Simulation. Version 1.0**, October 2010, www.functional-mockup-interface.org
- [10] **OPENPROD - Open Model-Driven Whole-ProductDevelopment and Simulation Environment**, www.openprod.org
- [11] Ch. Noll, T. Blochwitz, Th. Neidhold, Ch. Kehler: **Implementation of Modelisar Functional Mock-up Interfaces in SimulationX**. 8th International Modelica Conference. Dresden 2011.
- [12] J. Bastian, Ch. Clauß, S. Wolf, P. Schneider: **Master for Co-Simulation Using FMI**. 8th International Modelica Conference. Dresden 2011.

Master for Co-Simulation Using FMI

Jens Bastian Christoph Clauß Susann Wolf Peter Schneider
 Fraunhofer Institute for Integrated Circuits IIS / Design Automation Division EAS
 Zeunerstraße 38, 01069 Dresden, Germany
 {Jens.Bastian, Christoph.Clauss, Susann.Wolf, Peter.Schneider}@eas.iis.fraunhofer.de

Abstract

Co-Simulation is a general approach to simulate coupled technical systems. In a master-slave concept the slaves simulate sub-problems whereas the master is responsible for both coordinating the overall simulation as well as transferring data. To unify the interface between master and slave the FMI for Co-Simulation was developed. Using FMI a master was implemented with simple and advanced algorithms which can be applied depending on the properties of the involved slave simulators. The master was tested amongst others by coupling with SimulationX.

Keywords: co-simulation; FMI; master

1 Introduction

Modeling problems in natural sciences and engineering often leads to hybrid systems of differential and algebraic, time continuous and time or event discrete equations. Often complex multi-disciplinary systems cannot be modeled and simulated in one simulation tool alone or subsystem models are available only for a specific simulation tool. Sometimes sub-problems shall be simulated with the simulator which suits best for the specific domain. Thus for the simulation of multi-disciplinary problems or for hardware-in-the-loop simulation it is often reasonable or even necessary to couple different simulation tools with each other or with real world system components.

Simulator coupling is used in various fields of application like automotive engineering, microelectronics, mechatronics etc.

Up to now simulator coupling is nearly always a point-to-point solution tailored to the involved simulators. These special solutions cause high effort so a generally accepted interface for simulator coupling supported by many simulation tools is desirable.

2 Co-Simulation

Co-simulation is an approach for the joint simulation of models developed with different tools (tool coupling) where each tool treats one part of a modular coupled problem. Intermediate results (variables, status information) 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.

2.1 Coupling of simulators

A simulation tool S can be coupled if it is able to communicate data during simulation at certain time points t , cf. Figure 1. Here input variables are denoted by u and output variables by y .

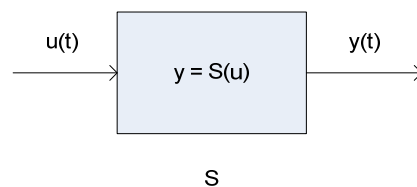


Figure 1: Block representation of a simulator S

Simulators have different capabilities which have an influence on the algorithms that can be used for their coupling. Such capabilities are:

- The simulator can handle variable communication step sizes.
- The simulator can handle events.
- It is possible to undo a time step, i.e. the simulator can reject time steps.

When using simulator coupling the original problem is divided into N subproblems each handled by a simulator. Typically, N is small, i.e. below 20. Thereby the simulators do not have to be different.

The signal flow for the coupled simulators can be described by a directed graph with the simulators as the nodes and the exchanged data as the edges.

If there is feedback in the graph then cycles exist. A cycle is a path in a graph with the same node as start and end point. Cycles can be eliminated if the simulators in a cycle are combined into a super-simulator i.e. a simulator superior to the simulators of the cycle.

Figure 2 shows an example of such a graph. Simulator A has the highest priority. The simulators B, C, and D form a cycle. E, F, and G are subordinated to this cycle. That means simulator A is executed first of all. Then the cycle of B, C, and D is finished. Afterward simulators E and F are executed whereat both simulators can be run in parallel. At last G is processed.

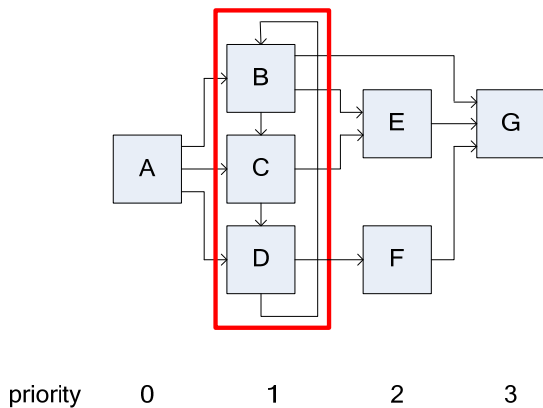


Figure 2: Example graph of coupled simulation tools

For simulation, the whole graph is analyzed first. If cycles are detected then they are combined into a super simulator. The simulators are coupled with directed data flow. A priority is assigned to each simulator with 0 representing the highest priority. Simulators with the same priority can be executed in parallel. All simulators in cycles either have to be processed iteratively or with small enough time steps and error control.

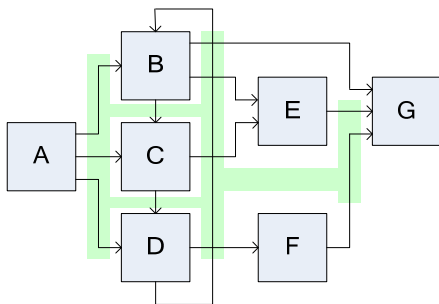


Figure 3: Master-Slave structure

Instead of direct coupling, a master is assumed to be located between the single simulation tools which

synchronizes, controls and manages them [1]. Each edge of the graph is regarded as to go “through” the master, cf. Figure 3. The master serves as an interface, establishes connections and exchanges data between the simulators which are called slaves. Slaves are assumed to communicate with the master only.

2.2 Basic Co-simulation computational flow

The whole co-simulation can be divided into several phases.

1. Initialization phase

All simulation tools are prepared for starting the co-simulation. The master receives the properties of the slaves. Furthermore the master receives the connection graph. The slaves and models are initialized and parameters are set. The communication links between master and slaves are established. The master chooses its algorithm based on the capabilities of the slaves as well as the connection graph and user input.

2. Simulation phase

The master forces the slaves to simulate the time interval from start time to stop time by stepwise solving master subintervals which are also called communication steps. Their boundaries are called communication points. In case of event iteration the communication step size can be zero. The simulation is performed independently for all subsystems restricting data exchange between subsystems to these communication points.

Before simulating a subinterval a slave receives its input values and possibly their derivatives with respect to time as well as the communication step size from the master. After finishing the communication step the master receives the output values of the slave and possibly their derivatives with respect to time. Furthermore the slave status has to be transferred to the master. If the slave simulation fails further communication is necessary.

3. Closing phase

The master stops the complete simulation and is responsible for proper memory deallocation, terminating and resetting or shutting down the slaves.

2.3 Accuracy and stability

Co-simulation can lead to problems regarding stability and accuracy of the simulation [2] – especially if feedback exists between simulators, cf. the example given in section 4.4. If a simulation tools provides an

interface for co-simulation at all then usually it is not possible to reset a simulator so that a time step can be repeated e.g. with a smaller step size.

So co-simulation should often be used as a last resort as long as iterative methods have to be used for stability and simulators only provide a rudimental co-simulation interface. Hopefully this will change in future with the introduction of a standardized co-simulation interface like the one proposed in the next section.

3 Functional Mock-up Interface (FMI) for Co-Simulation

The Functional Mock-up Interface (FMI) for Co-Simulation [3], [4], [5] is an interface standard for the solution of time dependent coupled systems consisting of subsystems that are continuous in time or time-discrete. It provides interfaces between master and slaves and addresses both data exchange and algorithmic issues. Both simple as well as more sophisticated master algorithms are supported. However, the master algorithm itself is not part of FMI for Co-Simulation.

FMI for Co-Simulation consists of two parts:

- *Co-Simulation Interface*: a set of C functions for controlling the slaves and for data exchange of input and output values as well as status information.
- *Co-Simulation Description Schema*: defines the structure and content of an XML file. This slave specific XML file contains “static” information about the model (input and output variables, parameters, ...) and the solver/simulator (capabilities, ...).

The complete interface description can be obtained from [3].

The capability flags in the XML file characterize the ability of the slave to support advanced master algorithms which use variable communication step sizes, higher order signal extrapolation etc.

A component implementing the FMI is called Functional Mock-up Unit (FMU). It consists of one zip file containing

- the XML description file and
- the implementation in source or binary form (dynamic library).

A master can import an FMU by first reading the model description XML file contained in the zip file.

Coupling simulators by FMI for Co-Simulation hides their implementation details and thus can protect intellectual property.

FMI for Co-Simulation version 1.0 was published in October 2010. Currently it is planned to combine FMI for Co-Simulation with FMI for Model Exchange to an FMI standard.

4 EAS Master

MODELISAR [6] is a research project within the European ITEA2 program. It is aimed to develop the FMI as well as to support it by involved tool vendors. Use cases will show the benefits of applied FMI. Master algorithms are not standardized with FMI but developed in the MODELISAR project e.g. by tool vendors. A prototypical implementation of a master has been provided by EAS for the MODELISAR consortium. The package contains the ANSI C code of the master, a generic “C function” slave, and a collection of examples.

The “C function” slave provides the basic functionality of FMI for Co-Simulation. The user has only to provide two functions for initialization (the number of input and output variables) and the computation of a step with the step size communicated by the master.

4.1 Configuration

The master is configured by a simple text file. There are keywords for start and stop time, step size, coupling algorithm, error tolerance etc. The coupled FMUs with their paths have to appear within the configuration file, too. The graph of the simulator coupling has to be supplied by an incidence matrix and information about the priority of the slaves as well as occurring cycles.

4.2 Coupling algorithms

The master prototype provides three algorithms for the simulation with fixed step size:

- data flow between the slaves without iterations, i.e. simple forward calculation
- fixed point iteration of all cycles within the graph
- simple implementation of Newton’s method with Jacobians approximated by finite differences

All master algorithms proceed in macro steps of fixed step size from start time to end time.

The computation of a time step from t_i to t_{i+1} within cycles is performed in the following way: Every slave makes an assumption for its input value u at time t_{i+1} . Currently this is done using constant interpolation $u(t_{i+1})=u(t_i)$, i.e. in each macro step

all terms that couple the subsystems are frozen. Thus synchronization and update of the exchanged values with computed output $y(t_{i+1})$ is done at the end of the time step. Because no slave depends on the current output of another one, the slaves can run in parallel. This iteration scheme is called to be of Jacobi type.

Another approach would be to simulate a time step with every slave of a cycle one after another and to use the output $y(t_{i+1})$ just calculated as input $u(t_{i+1})$ for the following slaves. These staggered algorithms which handle the subsystems sequentially are called of Gauß-Seidel type. This method was used within a first master implementation. The drawback of this approach is that the slaves within the cycles cannot run in parallel and the behavior of the iteration depends on the calling sequence of the slaves. However, an example exists where this approach converges while the first method does not converge.

4.3 Simple slave test examples

A collection of examples using the “C function” slave is provided together with the master. They cover different types of coupling – with or without cycles, nonlinear equations, ODEs, DAEs – and demonstrate the usage of the configuration file. Some of the examples can be solved with all master algorithms, some only with Newton’s method.

One of these examples is BspK6. It consists of four coupled slaves which exchange 4 values (0, 1, 2, 3) of type `fmiReal` and 2 values (4, 5) of type `fmiInteger`, cf. Figure 4. The slaves S0, S1, and S2 form a cycle.

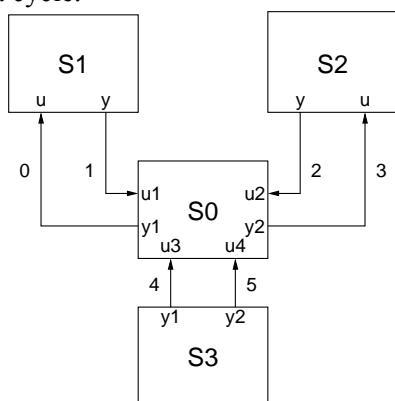


Figure 4: Example BspK6 from collection

Input, output, and internal variables of the slaves are related by the following equations.

Slave S0:

$$\frac{dx}{dt} + 2x - u_1 - u_2 = 0$$

$$y_1 := \begin{cases} 0 & u_4 = 1 \\ u_1 - x & \text{else} \end{cases}$$

$$y_2 := \begin{cases} 0 & u_3 = 1 \\ u_2 - x & \text{else} \end{cases}$$

Slave S1:

$$\frac{1}{2} \frac{dx}{dt} + x + u - \sin(3\pi t) = 0$$

$$(y - x) / (1000 \sin(\frac{2}{10} \pi t) + 1001) - u = 0$$

Slave S2:

$$\frac{1}{2} \frac{dx}{dt} + x + u - \sin(2\pi t) = 0$$

$$(y - x) / (-1000 \sin(\frac{2}{10} \pi t) + 1001) - u = 0$$

Slave S3:

$$y_1 = \begin{cases} 1 & \sin(\pi t) > \frac{1}{2} \\ 0 & \text{else} \end{cases}$$

$$y_2 = \begin{cases} 1 & \sin(2\pi t) < -\frac{1}{2} \\ 0 & \text{else} \end{cases}$$

Figure 5 and Figure 6 show simulation results for constant step size 10^{-4} and Newton’s method as iterative method for the cycle. The other two methods do not converge for this example.

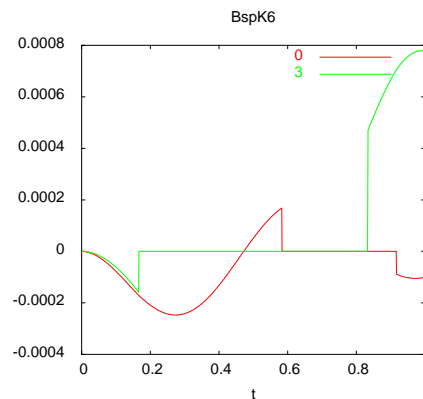


Figure 5: Simulation results for exchanged values 0 and 3

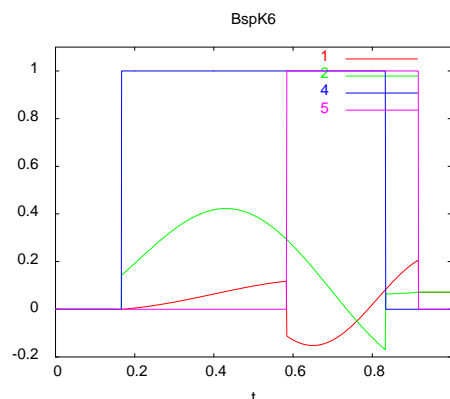


Figure 6: Simulation results for exchanged values 1, 2, 4, and 5

4.4 Coupling with ITI SimulationX

Another example shows coupling of SimulationX [7] with a “C function” slave via the EAS Master.

The original SimulationX model is shown in Figure 7. It is a simple plant with a controller driven by the “speed” function

$$f(t) = \begin{cases} 100 \text{ rpm} & 0.2 \text{ s} < t < 1 \text{ s} \\ 0 & \text{else} \end{cases}$$

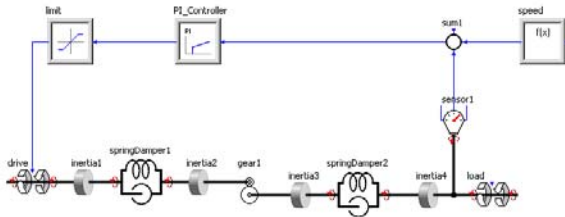


Figure 7: Full SimulationX model

This model has been split into three FMUs: two SimulationX FMUs for the controller and the plant and one “C function simulator” FMU for the speed input, cf. Figure 8. The SimulationX FMUs contain the model as well as the solver as a DLL. They were created via the code export option of SimulationX.

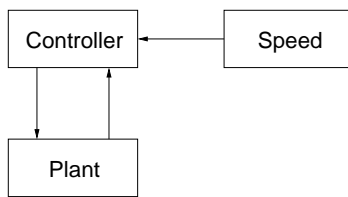


Figure 8: Coupling of three FMUs

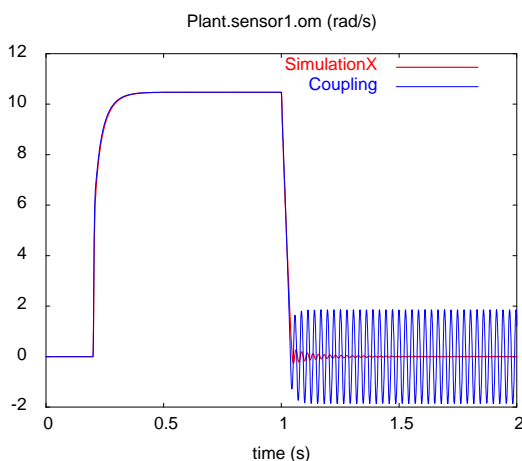


Figure 9: Simulation results

The coupled FMUs have been simulated by the prototypical master with fixed step size 10^{-3} with the simple algorithm for forward calculation without iteration. Results of this calculation as well as of the original simulation model are presented in Figure 9.

As it can be seen, the angular velocity of the plant shows a small but fast decaying oscillation in the original model after the speed has been switched to 0 after 1 s. In contrast, the oscillation is larger and does not decay in the simulator coupling. For larger step sizes the amplitude of this oscillation is even larger (not shown).

At the moment, SimulationX cannot discard steps so a simulation with iterative methods was not possible. With iterative methods we expect the oscillation to decay like in the original model.

4.5 Efficiency

Efficiency and simulation speed strongly depend on the problem which has to be solved.

Clearly, the most efficient approach would be to use only one simulation tool and do without co-simulation. If this is not possible then problems described by graphs without feedback can be simulated most efficiently using the non-iterative method. If there are cycles within the graph and no iterative methods can be used because the simulators cannot discard steps then accuracy and numerical stability may be poor. Anyway, the macro step size has to be very small then and thus the computational costs strongly increase.

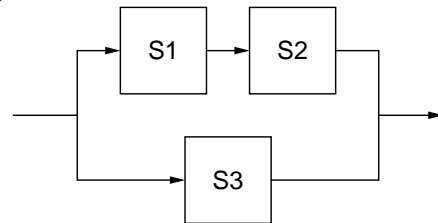


Figure 10: Disadvantage of current OpenMP approach compared to thread programming

By using OpenMP [8] slaves of the same priority can run in parallel. However, the current implementation of this approach has a disadvantage compared to thread programming which will be explained with the help of Figure 10. Here S1 has a higher priority than S2. S3 can have the same priority either as S1 or S2. Thus either both S3 and S1 can run in parallel and the simulation continues with S2 after both S1 and S3 have finished or S2 and S3 can both run in parallel after S1 has finished. Instead it would be better to handle S1 and S2 as a “super slave” which runs in parallel with S3, i.e. synchronization takes place at the start of S1 and S3 and after S2 and S3 have finished. However, either a more complicated data structure has to be used for this purpose if OpenMP should be used or platform dependent thread programming has to be used.

4.6 Summary of properties

The implementation of the EAS Master is as platform independent as possible. Platform dependent code – mainly for dealing with dynamic libraries – could happily be collected as preprocessor defines within a single header file. Thus the master runs on multiple platforms (MS Windows, Linux, Sun Solaris).

Slaves can run in parallel if they have the same priority. Platform independence also was the reason to use OpenMP instead of explicitly dealing with thread programming for this purpose. OpenMP is supported by newer version of the major C compilers (gcc, Visual Studio). Parallelization is realized by one `#pragma` directive in front of a “for” loop so that compilers without OpenMP support simply compile the code for serial execution. However, the OpenMP approach has the drawback compared to explicit dealing with threads that only slaves of the same priority and not across different priorities can run in parallel.

Currently the three algorithms mentioned in section 4.2 are available.

4.7 Future enhancements

A commercially available version of the master will have the following features:

- The graph will automatically be analyzed for the priority of the slaves and cycles.
- Newton’s method will be improved. A better Jacobian update strategy will be used so that the high cost of calculating a new Jacobian by finite differences will be reduced.
- Broyden’s method will be available as another iterative method.
- A step size control will be implemented based on results in [9] so that variable macro steps can be used.
- Polynomial interpolation of data besides the currently used constant interpolation will be supported.

5 Conclusions

Co-simulation is a powerful method to simulate heterogeneous systems where each subsystem is simulated by its own specialized simulator. However, currently simulation tools have their own interface for coupling – if at all. Additionally, they are often not able to discard steps and thus not suitable for iterative methods.

The Functional Mock-up Interface (FMI) for Co-Simulation as a proposed standard for simulator coupling will hopefully be widely used because it replaces current point-to-point solutions and thus eases the reuse of models tailored to special simulators. The protection of intellectual property is also possible with FMI.

Providing the prototypical master implementation will hopefully help to promote the FMI for Co-simulation.

Acknowledgements

The SimulationX model and FMUs were kindly provided by T. Blochwitz from ITI.

This work is supported by the German BMBF within the ITEA2 MODELISAR project.

The authors thank the reviewers for valuable remarks.

References

- [1] Wolf, S.; Blochwitz, T.: Master Slave Simulator Coupling. ITI Symposium 2010.
- [2] Schierz, T.; Arnold, M.: Advanced numerical methods for co-simulation algorithms in vehicle system dynamics. 1st Conference on Multiphysics Simulation, Bonn 2010.
- [3] Functional Mock-up Interface for Co-Simulation v1.0, MODELISAR consortium, 2010. <http://functional-mockup-interface.org>
- [4] Arnold, M.; Blochwitz, T.; Clauß, C.; Neidhold, T.; Schierz, T.; Wolf, S.: FMI-for-CoSimulation. 1st Conference on Multiphysics Simulation, Bonn, 2010.
- [5] Enge-Rosenblatt, O., Clauß, C.; Schneider, A.; Schneider, P.: Functional Digital Mock-up and the Functional Mock-up Interface – Two Complementary Approaches for a Comprehensive Investigation of Heterogeneous Systems. 8th International Modelica Conference, Dresden, 2011.
- [6] <http://www.modelisar.org>
- [7] <http://www.simulationx.com>
- [8] <http://www.openmp.org>
- [9] Schierz, T.; Arnold, M.; Eichberger, A.; Friedrich, M.: Study on Theoretical and Practical Aspects of Communication Stepsize Control. MODELISAR, sWP203 report, 2010.

An Annular Plate Model in Arbitrary Lagrangian-Eulerian Description for the DLR FlexibleBodies Library

Andreas Heckmann*, Stefan Hartweg* and Ingo Kaiser*

*German Aerospace Center (DLR), Institute of Robotics and Mechatronics
Oberpfaffenhofen, 82234 Wessling, Germany

Abstract

The bending deformation of rotating annular plates and the associated vibration behaviour is important in engineering applications which range from automotive or railway brake systems to discs that form essential components in turbomachinery.

In order to extend the capabilities of the DLR FlexibleBodies library for such use cases, a new Modelica class has been implemented which is based on the analytical description of an annular Kirchhoff plate. In addition the so-called Arbitrary Lagrangian-Eulerian (ALE) representation has been adopted so that rotating and non-rotating external loads may be applied conveniently to rotating plates.

Besides these particularities the new class AnnularPlate completely corresponds to the concept of FlexibleBodies library with the two already available model classes Beam and ModalBody.

This paper gives an overview on the theoretical background of the new class AnnularPlate, explains the usage and presents application examples.

Keywords: Arbitrary Lagrangian-Eulerian approach, annular Kirchhoff plate, flexible multibody system

1 Introduction

The commercial DLR FlexibleBodies library presented in 2006 [1] contains two different types of model classes: The Beam model employs an analytical description of the deformation field, while a general ModalBody model is defined in such a way that the dynamic behaviour of a body with an arbitrary geometrical shape can be simulated if suitable finite element data of the body exist.

The new model class AnnularPlate introduced in this paper is implemented in the same manner as it applies for the Beam model. The analytical description of an annular Kirchhoff plate has been used to define

the object-oriented data structure called "Standard Input Data of flexible bodies" (SID), see [2], which is the general base of all models in the DLR FlexibleBodies library.

Rotating discs are a very common structure type in mechanical engineering. But their modeling often has to cope with the difficulty to describe non-rotating forces acting on the disc such as the normal and friction forces at a disc brake. Usually this requires a contact formulation in order to evaluate which material point of the disc is in contact with the pad at the considered point in time.

Due to the so-called Arbitrary Lagrangian-Eulerian description it is possible to provide a standard Modelica multibody frame connector which is however not linked to a material point of the plate, but slides over the surface of the plate as it is given for the brake disc-pad contact point. No contact problem has to be formulated and solved and normal and friction forces are convenient to apply at this frame connector.

In addition the AnnularPlate model is capable of defining material-fixed points on the plate with frame connectors to which forces, other bodies such as unbalances, springs etc. may be attached in the usual way.

2 Theoretical Background

This section shortly summarizes well-known fundamentals on structural dynamics of annular plates and on multibody dynamics. The modeling approach of the FlexibleBodies library utilizes these fundamentals and will be introduced in Sec. 3.

2.1 Partial Differential Equation

The partial differential equation (PDE) of a freely vibrating, homogeneous Kirchhoff plate with transverse deformation w in the mid-plane of the plate as a function of the radius r , the angle ϕ and time t , i.e.

$w = w(r, \phi, t)$, reads

$$D \Delta \Delta w + \hat{\rho} \ddot{w} = 0, \quad (1)$$

where Δ denotes the Laplace operator and $\hat{\rho}$ represents the mass per unit area [3, (1.1)]. D is the bending rigidity of the plate according to

$$D := \frac{Eh^3}{12(1-\nu^2)} \quad (2)$$

and depends on the Young's modulus E , the plate thickness h and the Poisson number ν .

An analytical solution to (1) is assumed in the form

$$w(r, \phi, t) = R(r) \cdot \Phi(\phi) \cdot q(t), \quad (3)$$

so that the PDE (1) can be separated in three ordinary differential equations (ODE) for $R(r)$, $\Phi(\phi)$ and $q(t)$, respectively [4, 4.3.15].

For $R(r)$ the Bessel-type ODE

$$r^4 R'''' + 2r^3 R''' - (1 + 2k^2)(r^2 R'' - rR') + (k^4 - 4k^2 - \lambda^4 r^4)R = 0, \quad (')' := \frac{d}{dr}, \quad (4)$$

is obtained [5, (5.1-120)]. The parameter k relates (4) to $\Phi(\phi)$ in (3) and represents the wavenumber or the number of nodal diameters in Fig. 2. The parameter λ depends on the eigenvalue ω of the ODE for $q(t)$:

$$\lambda^2 := \omega \sqrt{\frac{\hat{\rho}}{D}}. \quad (5)$$

Bessel and modified Bessel functions of first and second kind satisfy (4) and have to be selected in such a way that the boundary conditions at the inner and the outer radius of the annular plate are considered.

Harmonic waves provide a solution with respect to the angular coordinate, i.e. $\Phi(\phi) = \cos(k\phi + \psi_k)$ with offset angle ψ_k and $\Phi(\phi) = \Phi(\phi + 2\pi)$.

Finally, the time-dependency of the displacements $q(t)$ are as well assumed to be harmonic, e.g. $q(t) = \sin(\omega t)$.

Note that (1) has an infinite number of solutions, out of which only a reduced, finite number of eigenvalues ω and associated deformation fields, the eigenforms in Fig. 2, are considered for numerical analysis.

2.2 Multibody Framework

The mechanical description in multibody dynamics is based on the floating frame of reference approach, i.e. the absolute position $\mathbf{r} = \mathbf{r}(\mathbf{c}, t)$ of a specific body particle is subdivided into three parts: the position vector

$\mathbf{r}_R = \mathbf{r}_R(t)$ to the body's reference frame, the initial position of the body particle within the body's reference frame, i.e. the Lagrange coordinate $\mathbf{c} \neq \mathbf{c}(t)$, and the elastic displacement $\mathbf{u}(\mathbf{c}, t)$:

$$\mathbf{r} = \mathbf{r}_R + \mathbf{c} + \mathbf{u}, \quad (6)$$

where all terms are resolved w.r.t. the body's floating frame of reference (R).

Therefore the angular velocity of the reference frame $\boldsymbol{\omega}_R$ have to be taken in account when the kinematic quantities velocity \mathbf{v} and acceleration \mathbf{a} of a particle are derived:

$$\mathbf{v} = \tilde{\boldsymbol{\omega}}_R \mathbf{r} + \dot{\mathbf{r}} = \mathbf{v}_R + \tilde{\boldsymbol{\omega}}_R (\mathbf{c} + \mathbf{u}) + \dot{\mathbf{u}}, \quad (7)$$

$$\mathbf{a} = \mathbf{a}_R + (\dot{\tilde{\boldsymbol{\omega}}}_R + \tilde{\boldsymbol{\omega}}_R \tilde{\boldsymbol{\omega}}_R) (\mathbf{c} + \mathbf{u}) + 2\tilde{\boldsymbol{\omega}}_R \dot{\mathbf{u}} + \ddot{\mathbf{u}}, \quad (8)$$

where the $(\tilde{})$ -operator is used to replace the vector cross product by a multiplication with an appropriate skew-symmetric matrix, so that e.g. the identity $\boldsymbol{\omega} \times \mathbf{c} = \tilde{\boldsymbol{\omega}} \mathbf{c}$ holds.

The decomposition in (6) makes it possible to superimpose a large non-linear overall motion of the reference frame with small elastic deformations.

The displacement field of the annular plate is approximated by a first order Taylor expansion with space-dependent mode shapes $\boldsymbol{\Phi}(\mathbf{c}) \in \mathbb{R}^{3,n}$ and time-dependent modal amplitudes $\mathbf{q}(t) \in \mathbb{R}^n$ [2]:

$$\mathbf{u} = \boldsymbol{\Phi} \mathbf{q}. \quad (9)$$

Note that the description of the annular plate is limited to this first order expansion in this initial implementation, so that plate buckling phenomena are not covered, see [6], [7, Ch. 1]. The second order displacement field of an annular plate currently is a field of research at the DLR.

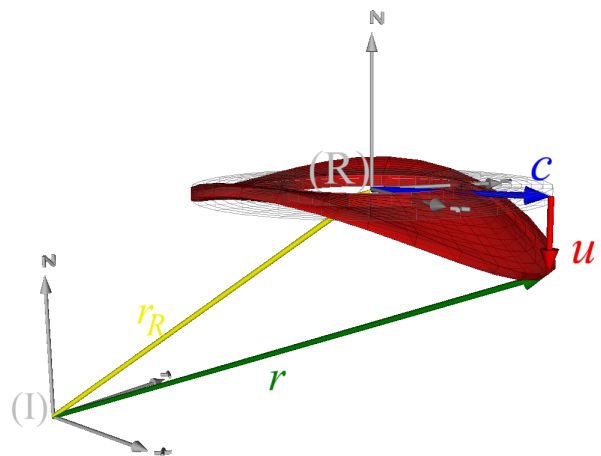


Figure 1: Vector chain to specify the position \mathbf{r} resolved in the floating frame of reference (R).

The number of considered mode shapes in the model is controlled by the values l_m and k_m , which are to be specified by user input parameters of the AnnularPlate class.

The functions $R_l(r)$ in (13) correspond to the Bessel functions mentioned in Sec. 2.1. However for the sake of simplicity and robustness of the implementation the original Bessel functions have not been used but a two-step approach is applied. At first the displacement field in radial direction and the underlying ODE (4) is discretized with cubic B-splines $\bar{\mathbf{R}}(r) = (\bar{R}_1(r), \bar{R}_2(r), \dots, \bar{R}_p(r))^T$ as shown in Fig. 4 taking the boundary conditions at the inner and outer radius into account [9].

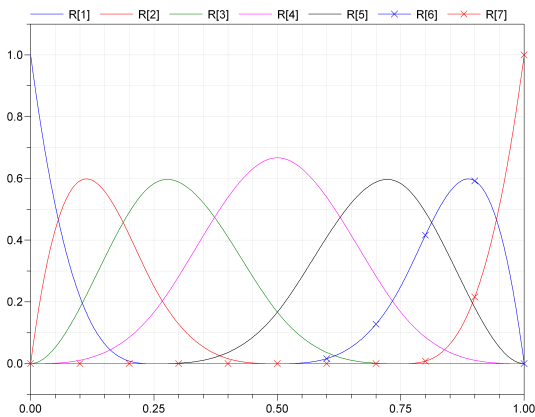


Figure 4: Example set of cubic B-splines for free boundary conditions to initially discretize the displacement field in radial direction.

That way the associated mass $\bar{\mathbf{M}}_e$ and stiffness matrix $\bar{\mathbf{K}}_e$ are evaluated. In the second step the problem

$$[\bar{\mathbf{M}}_e \omega_n^2 + \bar{\mathbf{K}}_e] \mathbf{v}_n = 0 \quad (14)$$

is solved for a specified number of eigenvalues ω_n . One specific eigenvector $\mathbf{v}_{n=l}$ may then be interpreted to define a fixed linear combination of the initial B-splines functions in such a way that the associated solution of (4) is approximated, i.e. $R_l \approx \mathbf{v}_l^T \cdot \bar{\mathbf{R}}$. The accuracy of the approximation may be controlled by the number of the initially used B-splines p in relation to the specified value l_m . The final result corresponds to the approach in (13).

3.2 Arbitrary Lagrangian-Eulerian Description

It is now considered that the annular plate performs a in general large rotation around its central axis specified by the angle $\chi(t)$. So far the motion of material

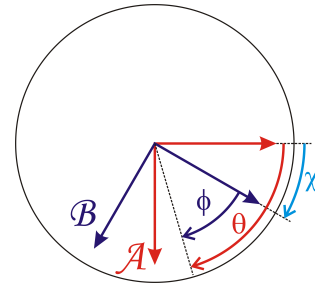


Figure 5: Coordinate transformation with angle χ , that leads from the Lagrangian to the ALE-description.

particles is described in the so-called Lagrangian point of view [10, Sec. I.3], i.e. the floating frame of reference follows the rotation as it is shown for the coordinate system named \mathcal{B} in Fig. 5.

However for specific use cases it may make sense to resolve the deformation of the plate in frame \mathcal{A} in Fig. 5 that follows the complete reference motion of the plate except of the motion expressed by the angle χ . In other words, the observer does not rotate with the plate but looks on the plate from the outside, from a point in rest concerning the rotation with angle $\chi(t)$.

This idea is influenced by the Eulerian description [10, Sec. I.4] widely used in fluid dynamics, where the motion state of the fluid at a fixed point in space is presented. However the concept introduced above combines aspects of the Lagrangian and the Eulerian approach and is therefore known as Arbitrary Lagrangian-Eulerian (ALE) description in literature, see e.g. [11]. Due to the rotational symmetry properties of the annular plate the ALE-description can here be formulated in an elegant way.

For theoretical derivation the coordinate transformation

$$\phi = \theta - \chi \quad (15)$$

is defined, where θ specifies the angular position of an observed point on the annular plate resolved with respect to the ALE-reference system \mathcal{A} in Fig. 5.

Furthermore it is assumed that for every mode shape in (13) that employs a $\sin(k\phi)$ -term an associated mode shape is present where the sinus- is replaced by the cosinus-function only, but $R_l(r)$ and k are identical, so that mode shape couples c_1 and c_2 exist:

$$\begin{aligned} c_1(r, \phi) &= R_l(r) \cdot \sin(k\phi), \\ c_2(r, \phi) &= R_l(r) \cdot \cos(k\phi). \end{aligned} \quad (16)$$

If the following identities

$$\begin{aligned} \sin(k\phi) &= \sin(k\theta) \cos(k\chi) - \cos(k\theta) \sin(k\chi), \\ \cos(k\phi) &= \cos(k\theta) \cos(k\chi) + \sin(k\theta) \sin(k\chi) \end{aligned} \quad (17)$$

are inserted into (16), an associated mode couple $\bar{c}_1(r, \theta)$ and $\bar{c}_2(r, \theta)$ defined with respect to frame \mathcal{A} appears:

$$\begin{aligned} c_1(r, \phi) &= \\ &= \underbrace{R_l \sin(k\theta)}_{:=\bar{c}_1(r, \theta)} \cos(k\chi) - \underbrace{R_l \cos(k\theta)}_{:=\bar{c}_2(r, \theta)} \sin(k\chi), \\ &= \bar{c}_1(r, \theta) \cos(k\chi) - \bar{c}_2(r, \theta) \sin(k\chi), \\ c_2(r, \phi) &= \bar{c}_1(r, \theta) \sin(k\chi) + \bar{c}_2(r, \theta) \cos(k\chi). \end{aligned} \quad (18)$$

As a result of suitable transformations it may also be written:

$$\begin{aligned} \bar{c}_1(r, \theta) &= c_2(r, \phi) \sin(k\chi) + c_1(r, \phi) \cos(k\chi), \\ \bar{c}_2(r, \theta) &= c_2(r, \phi) \cos(k\chi) - c_1(r, \phi) \sin(k\chi). \end{aligned} \quad (19)$$

The modes $\bar{c}_1(r, \theta)$ and $\bar{c}_2(r, \theta)$ are defined in the ALE-reference system \mathcal{A} and are linear combinations of the modes $c_1(r, \phi)$ and $c_2(r, \phi)$ described in the Lagrangian frame \mathcal{B} , whereas the combination depends on χ .

This information can be exploited in order to define a transformation: a deformation field resolved in the Lagrangian frame can be transformed to be resolved in the ALE frame and vice versa. Of course the physical deformation field itself does not change, but its resolution does so that the numerical values describing the deformation field will be different in frame \mathcal{A} or \mathcal{B} , respectively.

In practise the transformation is formulated in terms of the modal amplitudes $q_i(t)$ which are the deformation variables in (11):

$$\begin{aligned} \bar{q}_{i1}(t) &= \sin(k\chi(t)) \cdot q_{i2}(t) + \cos(k\chi(t)) \cdot q_{i1}(t), \\ \bar{q}_{i2}(t) &= \cos(k\chi(t)) \cdot q_{i2}(t) - \sin(k\chi(t)) \cdot q_{i1}(t). \end{aligned} \quad (20)$$

Again, the new modal amplitudes in the ALE frame $\bar{q}_i(t)$ are expressed as a linear combination of modal amplitudes in the Lagrangian frame $q_i(t)$ and it is just a matter of convenience and practicability in which coordinates the equations of motion are actually evaluated.

One particularity has been ignored so far. For mode shapes with $k = 0$, i.e. no nodal diameters in Fig. 2, no mode couple with c_1 and c_2 according to (16) exists, since no associated sinus-function is introduced in (13). As a consequence the transformation (20) is not defined for such modes. However, eigenforms with $k = 0$ present rotational-symmetric deformation fields since the dependency on ϕ is eliminated in (13) due to the term $\cos(k\phi)$. As a consequence eigenforms with $k = 0$ are invariant with respect to

rotations with angle χ or in other words: The modal coordinates $q_i(t)$ related to $k = 0$ are identical in the ALE- and the Lagrangian description and no transformation is needed.

4 The User Interface

4.1 Connectors and Parameters

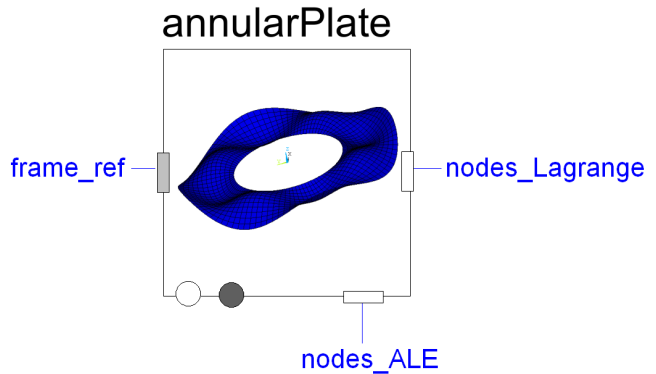


Figure 6: Icon layer of the AnnularPlate class with 3 types of multibody connectors: the floating frame of reference and two arrays of frames representing points in Lagrangian- or ALE-description, respectively.

Fig. 6 presents the AnnularPlate icon. Connections to the floating frame of reference of the plate are to be defined using the *frame_ref* connector. The array of connectors *nodes_Lagrange* contains as much frames as are given by the first dimension of the input parameter *ksi* in the following table:

geometrical parameters		
<i>r_i</i>	[m]	inner radius of the plate
<i>r_a</i>	[m]	outer radius of the plate
<i>th</i>	[m]	thickness of the plate
<i>ksi</i> [:, 2]	[–]	points on the disc

Each row of *ksi* specifies the radial and the angular position of one point in the mid-plane of the disc parametrized in the interval $[0, 1]$, e.g. $ksi[1, :] = \{0.5, 0.125\}$ defines a point in the middle between the inner and outer radius at 45° angular position.

The connector array *nodes_ALE* refers to the same input parameter definition *ksi*, whereas the associated points here are described in the ALE-representation. Forces and torques applied to these frames are in rest which respect to the rotation χ of the disc.

Note that conventional frame connectors from the Standard Multibody library are used within the Flexi-

bleBodies library and no restrictions concerning connecting e.g. other bodies to *nodes_ALE* are effected, although *nodes_ALE* frames do not represent material points.

Usually multibody frame connections represent physical mounting devices such as screws or welds that bond two frames together so that their positions and orientations are constrained to be identical. However it is the idea of *nodes_ALE* frames that they are not bonded to the disc and there is no mounting device. From the plate's material point of view *nodes_ALE* frames slide on the plate. In view of this fact the user is in charge to ensure that connections to *nodes_ALE* frames are physical consistent. If e.g. another body is attached to a *nodes_ALE* frame this would require a physical guidance device on the plate to which the external body is connected.

In addition to the 3-dimensional multibody frame connectors, two 1-dimensional rotational flanges are shown in Fig. 6. These two flanges are connected to both sides of the 3-dimensional rotational joint which is introduced into the *AnnularPlate* class at the plate axis by default. The two flanges are conditionally instantiated controlled via user parameter and can be utilized to e.g. define constant rotation velocity.

In addition to the purely **geometrical parameters** above, the table below shows the **physical parameters** the user has to provide in order to employ a *AnnularPlate* instance:

physical parameters		
ρ	$[\text{kg}/\text{m}^3]$	mass density
E	$[\text{N}/\text{m}^2]$	Youngs's modulus
G	$[\text{N}/\text{m}^2]$	Shear modulus

The following **discretization parameters** control the modal approach of the *AnnularPlate* model according to (13):

- *boundaryConditionRI*: This enumeration parameter offers the options *free*, *supported* and *clamped* and specifies the boundary condition at the inner radius.
- *boundaryConditionRA*: This is again an enumeration parameter that specifies the boundary condition at the outer radius in the same way as noted for the inner radius.
- *nodalDiameters*: This is an integer vector of arbitrary length, in which all nodal diameter numbers k , see Fig. 2, to consider have to be given. E.g. $\text{nodalDiameters} = \{0, 2\}$ defines

that all modes (to be additionally qualified by *nodalCircles*) with zero and two nodal diameters are to be taken into account.

- *nodalCircles*: This is an integer vector of arbitrary length, in which all nodal circles l to consider have to be given, see Fig. 2.
- *damping*: This is a real vector, which defines the damping of each mode separately.

There is one aspect in which the discretization parameters above differ from what is depicted in (13). There, the number of mode shapes is specified by two thresholds l_m and k_m and all modes with $l \leq l_m$ and $k \leq k_m$ are included in the model. However the two inputs *nodalCircles* and *nodalDiameters* offer the possibility to specify each nodal circle and diameter to be considered, separately.

A literature review had revealed that in particular brake squeal models often only include a single mode shape couple corresponding to a specific frequency at which squeal phenomena have been observed in real applications, see e.g. [12]. The case is covered by the parameter definitions above.

Besides the discretization parameters that are related to the underlying plate model the graphical user interface of the *AnnularPlate* class consists of a bundle of other input data to specify in-scale and exaggerated animation, initialization, state selection and so on. Concerning these more general issues the user interface corresponds to what is already known from the *Beam* and *ModalBody* class of the DLR *FlexibleBodies* library.

4.2 Degenerated Geometry

There are two different cases of degenerated geometry which lead to singularities if defined by user input:

circular plate: The *AnnularPlate* model is not capable of representing a true circular plate with $r_i = 0$. The model will simulate, if a very small inner radius such as e.g. $r_i = 1^{-10}\text{m}$ is given, but as long as not enforced by clamped boundary conditions the displacement results on the inner radius do not satisfy the compatibility equations of continuum mechanics, see [10, Sec. II.6]. E.g. consider two displacements $\mathbf{u}_A(r_i)$ and $\mathbf{u}_B(r_i)$ of two arbitrary, but not coinciding points lying on the inner radius, then the following statement has to be noticed in general:

$$\lim_{r_i \rightarrow 0} (\mathbf{u}_A(r_i) - \mathbf{u}_B(r_i))^2 \neq 0$$

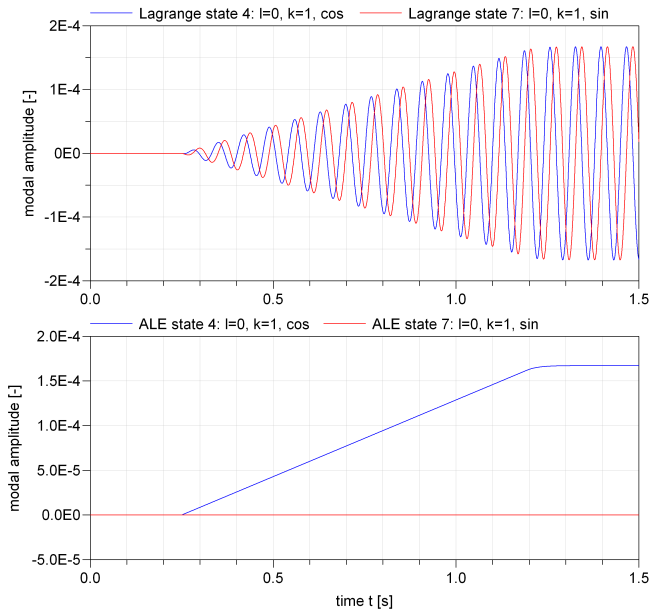


Figure 9: Modal amplitudes of two exemplary deformation states in Lagrangian and ALE-description.

is a known phenomenon that the discretization with eigenmodes is comparable inefficient whenever static deformation fields are to represent, so that a large number of modes is necessary to get correct values. Note that this fact does not apply for dynamic excitations.

Secondly, the application of a single, discrete force at the circumference of an annular plate is an issue for the angular discretization, which here may be interpreted as a Fourier expansion. Again, it is to expect that a large number of modes is necessary to get values close to reality.

A closer look at the exaggerated compared to the in-scale animation in Fig. 8 shows that deformations also occur in regions e.g. opposite to the force attachment point. These displacement results far away from the cutting tool are reduced if a higher number of nodal

Natural frequencies [Hz]					
Modelica	1449	1478	1478	1635	1635
Ansys	1451	1480	1480	1637	1637
Modelica	2064	2064	2847	2847	3974
Ansys	2065	2065	2848	2848	3977

Table 1: The first 10 natural frequencies of an AnnularPlate model compared to an Ansys model of the same plate with 1296 Shell63 elements for the sake of verification.

diameters k is considered.

The convergence of the deformation results as a function of the nodal diameters k and the nodal circles l is presented in Fig. 10, where the deformation at the force attachment points are given. At least the nodal

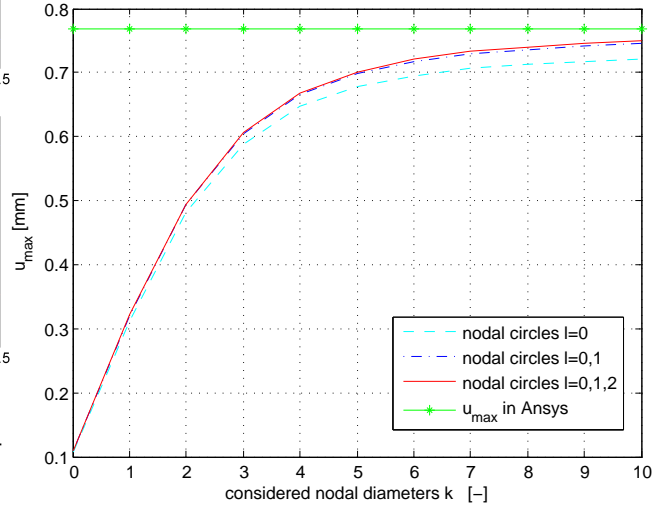


Figure 10: Convergence of the displacements results at the force attachment point

diameters $k = 0, 1, \dots, 7$ and the nodal circles $l = 0, 1$ should be taken into account in order to get reasonable static deformation results which leads to all together 30 degrees of freedom. The highest frequency in the model associated to the eigenform with $k = 7, l = 1$ turns out to be 18837 Hz. In the first implementation 44.7 cpu-s were required to simulate the 5.5 s of the complete scenario with a common lap-top.

5.2 A Helicopter Blade Control Model

The cyclic blade control of an helicopter is the second application example shown in Fig. 11. The swash plate, here modeled as an annular plate, supports two linkages that actuate the pitch joint of the helicopter blades. As long as the swash plate rotates in parallel to the rotor base carrying the blades above, the pitch of the blades is kept constant during one rotation. If the swash plate is tilted in such a way that the angular velocity vectors of the rotor base and the swash plate are no more collinear, the blade pitches are a function of the rotation angle, see Fig. 12.

Since the direction and value of the air forces acting on the blades depend on the pitch angle, the roll and pitch motion of the helicopter fuselage can be controlled via this mechanism.

Besides the swash plate, the linkages, the rotor base and the pitch joints, the model in Fig. 13 contains

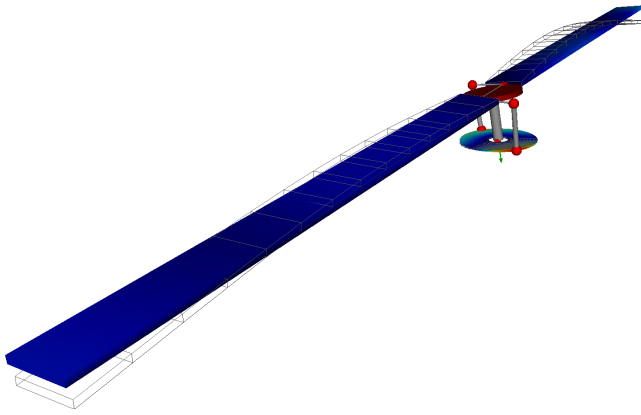


Figure 11: Total view on the helicopter mechanism: the wireframe illustrations exaggerate the deformations by a factor of 100, while the solid representations are shown in true scale.

two 5 m long beams describing the blades and considering their torsional and two-directional bending deformation. A rough representation of the air forces' effects on the pitch motion is given by force-damper elements acting on the pitch joints. The prismatic joint in Fig. 13 allows for the adjustment of the vertical position of the swash plate and thereby governs the collective pitch angle. The rotational joint aside regulates the tilting angle of the swash plate and therefore provides cyclic blade control.

The inner and outer radius of the 0.01 m thick swash plate made of steel is set to 0.1 and 0.39 m, respectively. Supported boundary conditions are applied at the inner radius and the $l = 0, k = 0, 1, \dots, 7$ eigenforms are considered, so that 15 degrees of freedom and eigenfrequencies between 50 Hz and 1009 Hz are defined. Since the external loads acting on the plate here rotate with the plate the ALE-functionality was

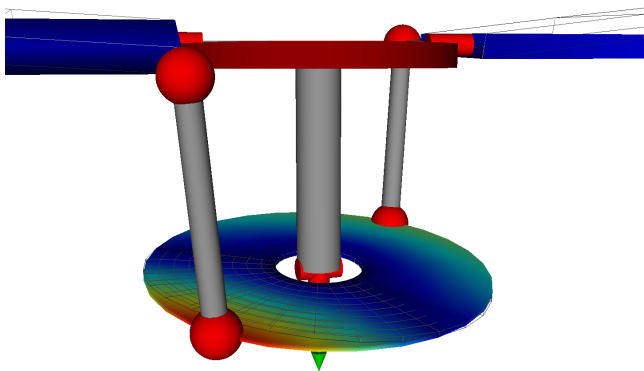


Figure 12: Side View on the tilted Swash Plate: the absolute value of the deformations are additionally indicated by color.

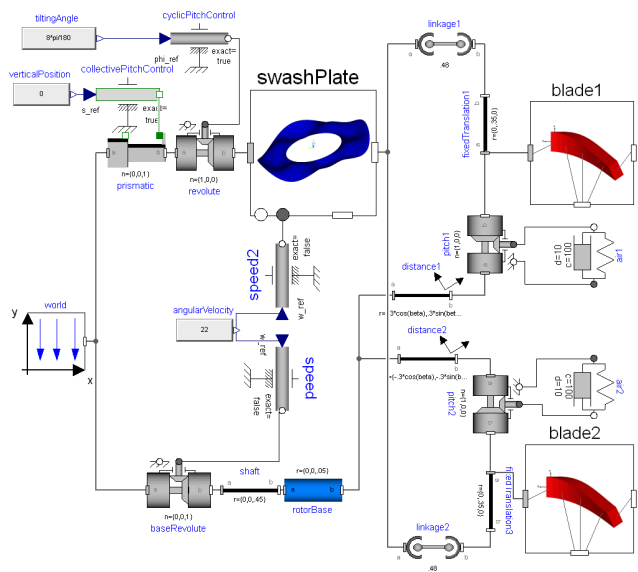


Figure 13: Diagram layer of a Helicopter Blade Control Model

not used. The blade models take the first eigenform for each of the three deformations types into account. In order to evaluate 1 s simulation time, 4.7 cpu-s are required on a common lap-top.

The simulation scenario assumes a constant angular motion of the blades with 22 rad/s, the swash plate tilting angle is as well constant, namely 8° . Fig. 14 shows the controlled pitch angles as a function of time.

The above plot in Fig. 15 presents the bending deformation of the plate at those two positions where the linkages are attached. Since the model is initialized in the undeformed configuration, natural vibration are initiated but are damped out rather quickly due to the defined structural damping of 2%.

The first modal amplitude in the plot below in Fig. 15 is associated to the rotational-symmetric $\langle k = 0, l = 0 \rangle$ -eigenform and its stationary value $q_1 \neq 0$ is ruled by the gravity load. The modal amplitude q_2

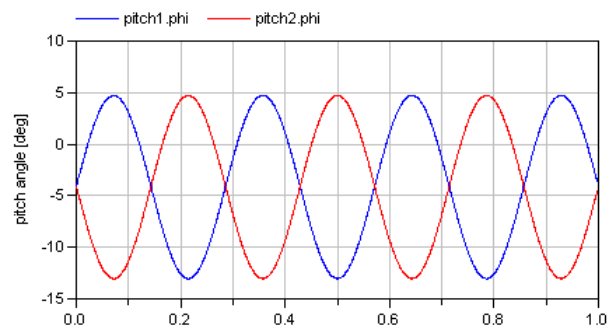


Figure 14: Simulation results concerning the controlled helicopter blade pitches.

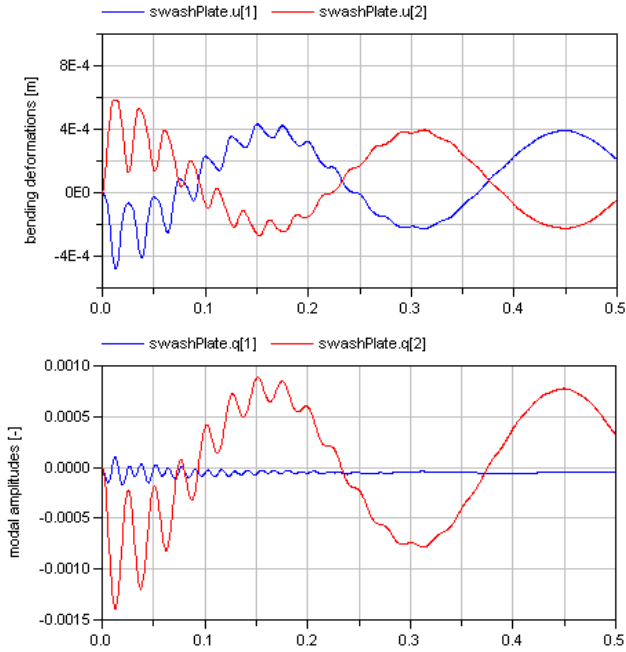


Figure 15: Deformation results at the two linkage attachment points and two modal amplitudes of the swash plate.

is related to that $\langle k = 1, l = 0 \rangle$ -eigenform, which displays its maximum and minimum deformations exactly at the linkage attachment points. As a consequence q_2 represents by far the dominating part of the particular solution.

5.3 A Brake Squeal Model

The last application is a reproduction of a brake squeal model presented by Chakraborty et al. [12]. It is based on the idea that the friction forces are oriented along

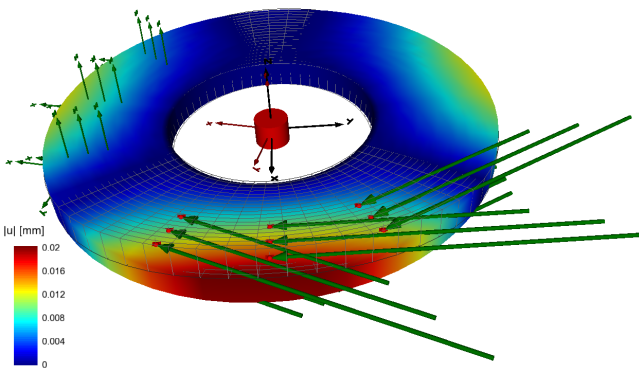


Figure 16: Animation of the brake disc with 18 applied friction forces oriented along the deformed surface (wireframe scale 1000:1). Pads and caliper bodies are considered but omitted for the visualization only.

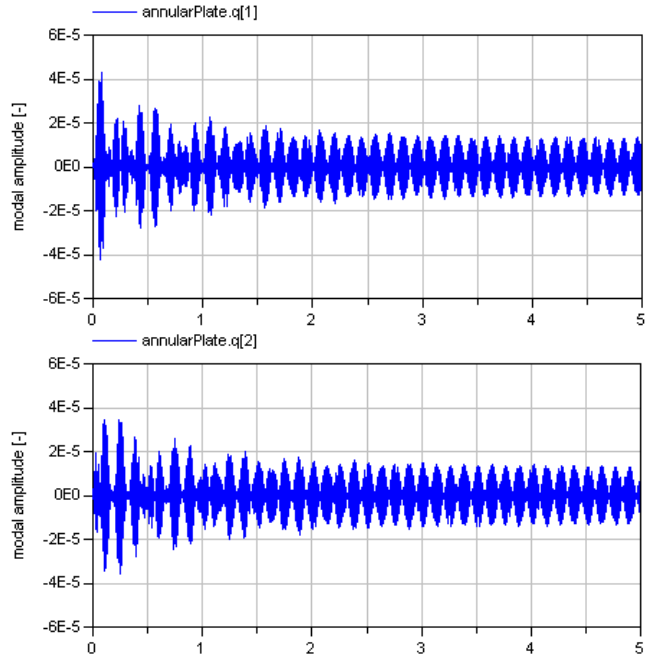


Figure 17: Modal amplitudes of the Brake Squeal Model.

the deformed friction surface. This so-called follower-forces phenomenon leads to a flutter-type instability and as a consequence to brake squeal. The arrows in the animation Fig. 16 show the friction forces as they are aligned with the surface tangent at the contact points. Due to this set-up the limit cycles in Fig. 17 occur as soon as the friction coefficient exceeds a certain limit.

The simulation scenario was defined as an initial value problem. Therefore the modal amplitudes of the first 1.5 s in Fig. 17 slightly differ from the behavior later on. The angular velocity of the brake disc was assumed to be constant 25 rad/s, the brake disc dimensions were set to $r_i = 0.07$ m, $r_a = 0.153$ m and $t_h = 0.0181$ m and 4 eigenforms with $l = 0, k = 1, 2$ with supported boundary conditions at the inner radius are considered. 64 cpu-s were needed to simulate the 5 s to be seen in Fig. 17.

The contact is formulated with one prismatic joint in axial direction for each contact point, see Fig. 18. The spherical joint allows for the alignment of the friction force with the contact surface. frame_b is to be connected to one nodes_ALE frame of the annular plate, see Fig. 6. frame_a is supposed to provide the connection to the brake caliper, which is a part of the model but not visualized in Fig. 16. The spring-damper element attached to the prismatic joint represents the contact stiffness. For a more advanced study, this linear element may be replaced by a non-

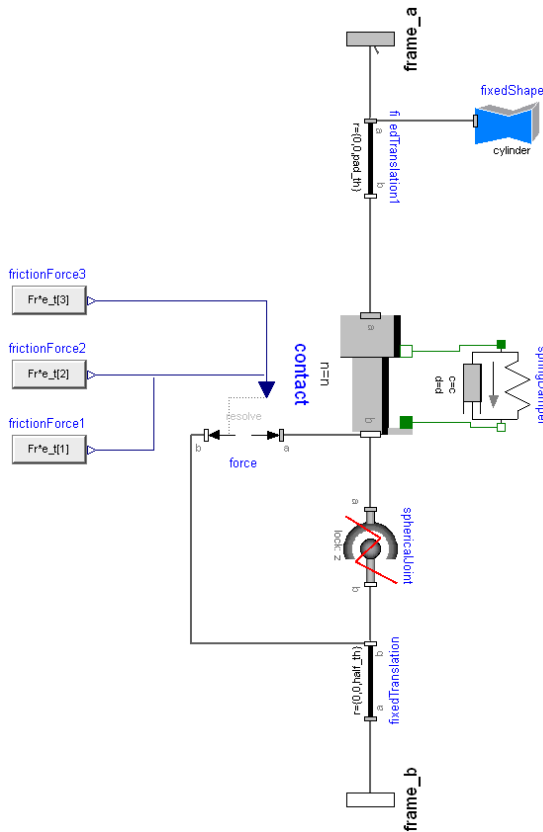


Figure 18: Diagram layer of the contact submodel.

linear spring which takes the loss of contact or the lift-off of the brake pads, respectively, into account. The simplicity of the contact modeling here again demonstrates the advantages of the ALE-description.

6 Conclusions and Outlook

This paper introduces the new Modelica class called AnnularPlate. The underlying plate model refers to a homogeneous Kirchhoff plate in cylindrical coordinates. The option to use connector frames in the so-called Arbitrary Lagrangian-Eulerian description offers the capability to apply non-rotating external loads in a convenient and numerical efficient way. The first example, a lathe cutting model, demonstrates in particular the advantages of this ALE-approach. The Helicopter Blade Control model presents the annular plate model as a part of a more complex mechanism. A Brake Squeal Model from literature concludes the example presentation. The AnnularPlate class will be distributed with the Version 2.0 of the commercial DLR FlexibleBodies library.

Future enhancements concern the second order displacement field description to cover initial plate buckling phenomena as well. The additional consider-

ation of torsional deformations of the plate is another optional improvement in order to cope with applications in which large forces in circumferential direction are present.

7 Acknowledgements

A first preliminary version of the annular plate model was implemented by Kemal Çiğ in the course of his master thesis project at the DLR.

This work is part of the ITEA 2 ~ 6020 project Eurosyslib and therefore funded by the German Federal Ministry of Economics and Technology. The authors highly appreciate the partial financial support of DLR by BMBF (BMBF Förderkennzeichen: 01IS07022F), the German Federal Ministry of the Education and Research, within the ITEA 2 project Eurosyslib [13].

References

- [1] Andreas Heckmann, Martin Otter, Stefan Dietz, and José Díaz López. The DLR FlexibleBody library to model large motions of beams and of flexible bodies exported from finite element programs. In *5th International Modelica Conference*, pages 85 – 95, 2006.
- [2] 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.
- [3] A.W. Leissa. *Vibration of plates*. NASA SP-160, Washington, D.C., 1969.
- [4] R. Szilard. *Theory and Analysis of Plates*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1974.
- [5] H. Irretier. *Grundlagen der Schwingungstechnik 2*. Vieweg-Verlag, Braunschweig, 2001.
- [6] O. Wallrapp and R. Schwertassek. Representation of geometric stiffening in multibody system simulation. *International Journal for Numerical Methods in Engineering*, 32:1833–1850, 1991.
- [7] F. Bloom and D. Coffin. *Handbook of Thin Plate Buckling and Postbuckling*. Chapman & Hall/CRC, Washington, D.C., 2001.

- [8] A.A. Shabana. *Dynamics of Multibody Systems*. Cambridge University Press, Cambridge, 2nd edition, 1998.
- [9] Carl de Boor. *A practical Guide to Splines*. Springer-Verlag, Berlin, 1978.
- [10] J. Salençon. *Handbook of Continuum Mechanics*. Springer-Verlag, Berlin, 2001.
- [11] U. Nackenhorst. The ALE-formulation of bodies in rolling contact, Theoretical foundations and finite element approach. *Computer Methods in Applied Mechanics and Engineering*, 193:4299–4322, 2004.
- [12] G. Chakraborty, T. Jearsiripongkul, U. von Wagner, and P. Hagedorn. A new model for a floating caliper disc-brake and active squeal control. *VDI-Bericht*, 1736:93–102, 2002.
- [13] ITEA 2. Eurosyslib. <http://www.itea2.org>.

A Modelica-Based and Domain-Specific Framework for Electromechanical System Design

Damien Chapon*, Fabien Hospital[†], Guillaume Bouchez*, Marc Budinger[†]

*Airbus Operation S.A.S.,

316 Route de Bayonne 31060 Toulouse,

[†]INSA de Toulouse

135 Av. de Rangueil 31500 Toulouse.

{damien.chapon,guillaume.bouchez}@airbus.com,

{fabien.hospital,marc.budinger}@insa-toulouse.fr

Abstract

A Modelica-Based Domain-Specific Framework for Electromechanical System Design was developed. The intended goal of this framework is to be used in early design phases in order to size physical architectures of electromechanical airbrake system. It has been developed using a generic methodology for the development of interoperable and model-driven system design frameworks. It is based on domain-specific modelling languages for the description of system architectures and relies on ModelicaML, a Modelica UML profile, to support system architecture analyses with the Modelica modelling language. Transitions between architectural description models and Modelica analysis models are realized through analyses-based model transformations.

Keywords: Modelica; Domain Specific Language; Model Driven Engineering, Electromechanical Actuator.

1 Introduction

To develop new generations of aircrafts which ensure safer flights with improved operations, new system architectures which encompass new technologies shall be developed. Moreover, aircraft development shall be realized in a shorter period and with a new complex industrial organization that enforces the links with the system suppliers. To face up to these technical and industrial challenges, more and more modelling and simulation are used during the aircraft development, from the preliminary and conceptual phase to the integration of systems, and at different levels from aircraft functional level, to

detailed dynamical analyses of equipments. However, the use of modelling and simulation activities within such compartmented and distributed organization results in the application of several different and non-fully coordinated or optimized “model-driven” processes, methods, and tools to support the discipline of systems engineering.

In order to solve this problem, a generic methodology for the development of interoperable and model-driven system design frameworks has been created. In this paper we are not going to present the overall methodology, but rather its philosophy, and how Modelica is integrated and used in it. For the demonstration, we applied this methodology in order to develop a Modelica-based domain-specific framework for electromechanical system design. The intended goal of this framework is to be used in early design phases in order to size physical architectures of electromechanical airbrake system. This Framework is integrated in an Eclipse platform. It is based on domain specific modelling languages for the description of system architectures and relies on ModelicaML, a Modelica UML profile, to support system analyses with the Modelica modelling language. Transitions between architecture descriptive models and Modelica analysis models are realized through analyses-based model transformations.

2 Design Framework development methodology overview

2.1 Methodology’s principles for collaborative and interoperable design activities

The design of a system physical architecture is an iterative process. It involves several interrelated sub processes to transform the system functional

architecture into a physical solution. The arrangement of these different sub processes are depicted in the following picture inspired from the IEEE 1220 standard [1].

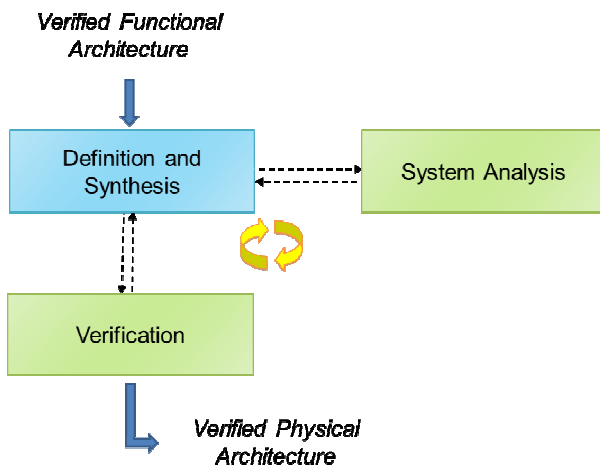


Figure 1 - System Physical Architecture design sub-processes

During the *Definition and Synthesis* sub-process, different alternatives of physical architecture are proposed and defined. The *system analysis* sub-process is used in support to evaluate and compare these proposed architectures. The selected architecture is then more precisely defined, optimised and sized in the *Definition and Synthesis* sub-process. Finally the *Verification* sub-process is used to verify and validate the chosen architecture in each of these domains of use. As illustrated in the Figure 1, the system physical architecture design process is an iterative set of back and forth between these different sub-processes.

Inside a system design team, the different activities of these sub-processes are distributed across different actors having different skills and roles. Depending on the system complexity and the organization these roles could be merged and their appellations could be different. However for complex systems these roles are clearly separated and it is therefore conceptually important to keep these roles separated in order to build optimized system design framework. To schematize, we can group these roles in three main categories:

- **System Architects.** They manage the overall system design activities. They define the different architecture alternatives. They ensure the consistence with the functional architecture and with other systems. They manage the detailed design and validate the choices and the results of the system designers and system analysts.
- **System Designers.** They define the architecture according concepts expressed by the system

- architect. They realize the detailed design, the optimization and sizing of the architecture;
- **System Analysts.** They realize the trade studies between the different architecture alternatives and/or the verification/validation analyses.

The different actors of the system design activities have to interact with the central point of all these activities, i.e. the system physical architecture being designed. Firstly to get the information that they need for the activities they are responsible for, and then to send back the results to the architecture definition after performing their activities. In order to get a collaborative design it is therefore crucial to give a central role to the system architecture and to allow the different actors accessing to the architectural data they need. In a classical document-centric process, the architectural data is stored in documents and is therefore not formalized and has to be interpreted by the different actors to realize their activities. Thus, with the growing importance of models and model-driven technologies in the system engineering processes, it could be useful to have a formal representation of this central architecture description in order to automate the access to the architectural data for the satellites design or analysis activities.

The Figure 3 illustrates the modular organization of the frameworks in order to get a collaborative design with dedicated environment for the different actors.

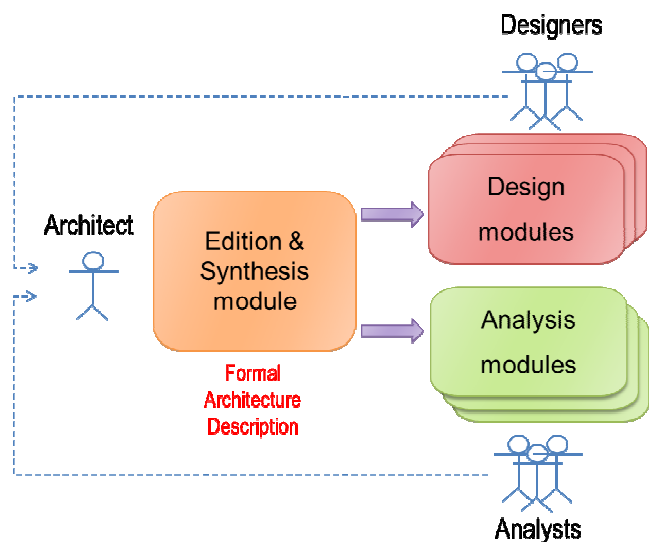


Figure 2 - Actors and modules interactions

Moreover, in order to get a functional and multi-system optimisation of the designed aircraft, the different teams responsible of systems architectures have to access to the architectural data of the others

systems their own system is related with. The architectural data is stored in the system architecture description, which should also be the link between the different teams. To support this data exchange we can imagine a common architectural database where every team could access to the others systems architectures descriptions through its own dedicated framework. The development of this database is out of the scope of the methodology. However this idea illustrates the need to build interoperable system design frameworks. We illustrated this idea in the Figure 3. This figure shows two dedicated design frameworks for two different system design teams. Each framework got a multi-view architectural editor for edition and consultation of system architectural data. Each team can access and edit the data of its system architecture through its own framework. Architectural data coming from others system can only be consulted. The consulting and editing views are therefore different for each team depending on their different representation needs. The design and analyses modules are plugged to the formal architectural data to access the data in order for the system designer or analyst to perform their activities. Then the results of these activities are used to mature the system architecture.

- Creation of architectural multi-views editors with dedicated Domain Specific Languages ;
- Development of gateway from system architecture description to design or analysis activities through model transformations.

With Domain-Specific Languages every actor of every system design activities could have access to the system architecture description in a view adapted to its needs. The creation of graphical Domain Specific Languages is composed of two main sub activities:

1. Creation and customization of domain specific languages with their own meta-models to capture the right knowledge and concerns of specific engineering domains;
2. Creation of customized graphical modelling editors. These graphical modelling editors bring the views that are needed by system architects to graphically create models conform to their engineering domain's meta-model.

Then, model transformations could be used to automate the exchange of data from architecture descriptive models to analyses models. If these satellite activities are not model-based, others models transformations such as automatic code generation or document generation are used to help the realization of these activities.

2.2 Model-driven development methods

The framework development methodology is model-driven. It concerns the two following main activities:

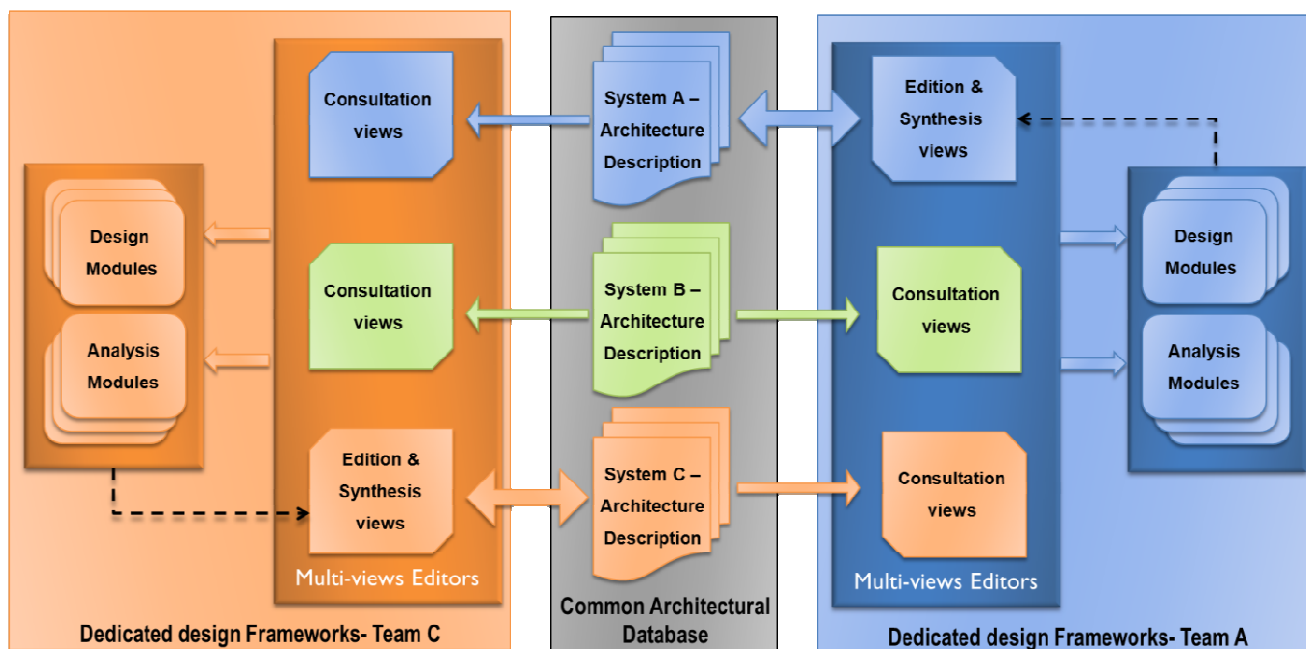


Figure 3 - Example of possible relationships between dedicated design frameworks through a Common Architectural Database

2.3 Model-driven technologies

Thus, the methodology relies on the creation of domain specific modelling languages in order to capture the specificities of engineering domains' knowledge. In this perspective it uses the Eclipse Modelling Framework (EMF) [2]. EMF is an Eclipse-based modelling framework and code generation facility for building tools and other applications based on a structured data model that can be specified in the Ecore language. Ecore is a variant of the EMOF, a subset of the OMG's MOF [3] standard, that is used to define simple meta-models using simple concepts. Moreover, EMF provides the foundation for interoperability with other EMF-based tools and applications. Interoperability between tools is a key driver in our methodology, so we selected ATL[4] (ATLAS Transformation Language), a model transformation language and toolkit, for the model transformations inside the use case presented in the next section. We selected Acceleo[5] for model to text transformation. For the creation of graphical editors we selected Obeo Designer [6]. Obeo Designer is an open workbench fully integrated with Eclipse. It is based on GMF [7] (Graphical Modelling Framework), that provides a generative component and runtime infrastructure for developing graphical editors based on EMF. Obeo Designer hides the complexity of GMF and offers the capacity to build quickly and easily customized graphical editors.

3 Use Case

3.1 Airbrake system presentation

In the present case study, an electromechanical actuator equivalent to a currently operating hydraulic airbrake actuator of a commercial single aisle aircraft is studied (Figure 4). The kinematics of the electromechanical airbrake is assumed to remain identical to the hydraulic one. This kinematics is based on a three rod mechanism, where the extension/retraction of the actuator linear jack drives the angular movement of the airbrake control surface. Accordingly, the transformations of motion are rotation (motor, gear), translation (screw, nut) and rotation (of the control surface). The airbrake motion ranges from 0 °to 50°. The moment of inertia of the airbrake introduces dynamic efforts that are not significant compared to the aerodynamic efforts. Specification imposes matching the dimensions of the current hydraulic actuator. Therefore, the variation of hard point positions was not addressed in this study. Furthermore, in power sizing phase,

these points will be considered as perfect mechanical transmissions (e.g., no friction and no backlash). But with the sizing progress, tools are more and more detailed, and friction, stiffness and backlash will be taken into account to implement the virtual prototype.

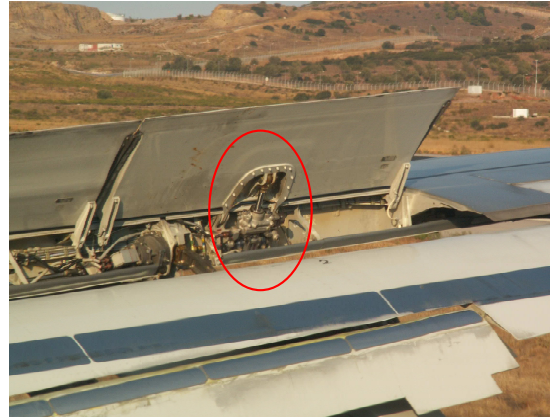


Figure 4 - Airbrake and its hydraulic actuator

3.2 Overview of the developed Domain-Specific Design Framework

For this use case, the development methodology introduced in section 2 has been used. A Modelica-Based Domain-Specific Framework for Electromechanical Actuators (EMAs) System Design was developed. The Figure 5 gives an overview of this Framework.

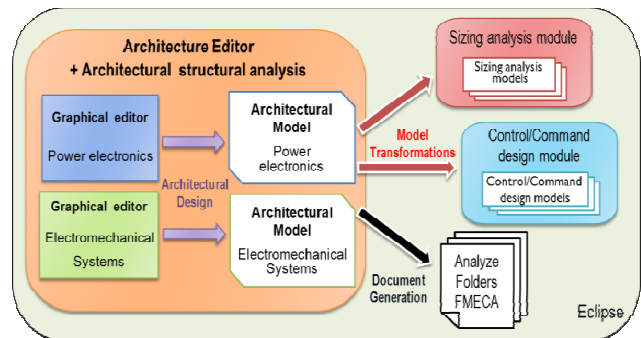


Figure 5 - General overview of the developed system design framework

As illustrated in the figure, the framework is composed of two system architecture graphical editors, one for each of the main domains collaborating for EMAs system design activities:

1. Power Electronics Systems, to control and make the correct conversion of the electric power coming from the electrical network to the electromechanical motor;

2. Electromechanical Systems, to transform the electrical power in mechanical power and to adapt the mechanical power to the application.

Then, model transformations and document generation capabilities have been developed to support partially or totally the following activities:

- Parametrical and structural analyses;
- Sizing analyses;
- Failure Mode, Effects, and Criticality Analyses (FMECA);
- Airbrake position control synthesis.

3.3 Domain Specific Modelling Languages for Electromechanical System Design

For the development of the two system architecture graphical editors, two meta-models were developed, one for electromechanical actuating system architecture and one for power electronics system architecture. Then, the domain specific graphical editors for each domain have been realised in Obeo Designer so that the system architects can build graphically the architectural models. In Obeo Designer we specified the graphical representations of each required concepts (components, ports and connections) of the two meta-models. We assigned a domain specific icon to each component and used generic graphical representations for ports and connections. Then we created the palette of components and connections, and we specified the way the model elements are created when using the palette. The figure 6 and 7 presents two architectures that have been realized with these graphical editors.

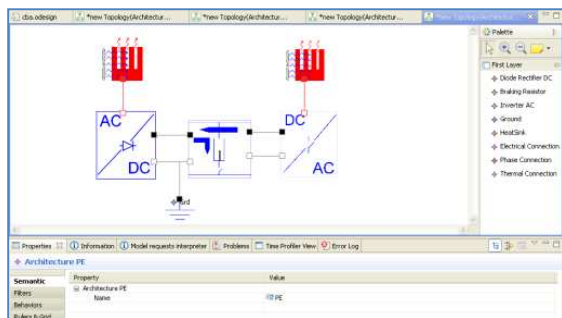


Figure 6 - Power electronics system architecture graphical editor

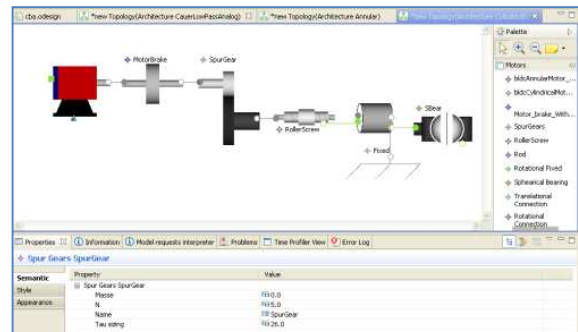


Figure 7 – Electromechanical system architecture graphical editor

As can be seen, the graphical editors propose a palette of components and connections that can be disposed on the workbench. The components' attributes can be changed in the properties view. Obeo Designer keep updated instantaneously the graphical view of the model and the model itself.

3.4 Integration of ModelicaML

Modelica [8] is a multi-domain modelling language for efficient component-oriented modelling of complex systems. The Modelica formalism can be used by several domains to perform physical analyses. Modelica is therefore well suited for multi-domain physical analyses and consequently we add it as an analyses module in order to add virtual analyses capabilities. The link between system architecture descriptive models and Modelica analyses models as already been studied in a previous work [9] and this integration is the concretisation of this work.

The Modelica module used in the framework is the ModelicaML [10] eclipse plug-in. Actually ModelicaML is a UML [11] profile. It extends a subset of UML in order to graphically define new Modelica models by using UML diagrams. These UML diagrams allow presenting the composition, connection, inheritance or behaviour of classes. Thus, it brings Modelica modelling capabilities into the framework. Further it relies on the OMG's UML, which is conform to the Meta-Object-Facility and therefore the model transformations between the system architecture descriptive models and the analyses models can be easily defined in ATL.

3.5 Analysis-based model transformations to Modelica Model

With the integration of a ModelicaML analyses module, the architecture descriptive models can be used as inputs for model transformations in order to create Modelica analyses models. In the system

engineering process, analyses are performed for specific purposes and therefore domain experts use adapted modelling languages to create the adequate analyses models.. And finally the domain experts dispose and connect the required components according the known system architecture to be analysed. In the same way, the developed framework relies on model transformations in order to automate as much as possible these analyses models creation steps.

According to the analysis to be performed, a specific ATL model transformation is coded. The Model transformation is performed in three steps as illustrated in Figure 8:

1. Components of the architecture to be analysed are directly mapped to the right analyses components available in a library dedicated to the analysis to be performed.
2. Parameters of each component of the architecture are mapped to the right parameters of their corresponding analyses models.
3. Connections of the architecture models are treated by the model transformation to automatically connect the components of the analyses models.

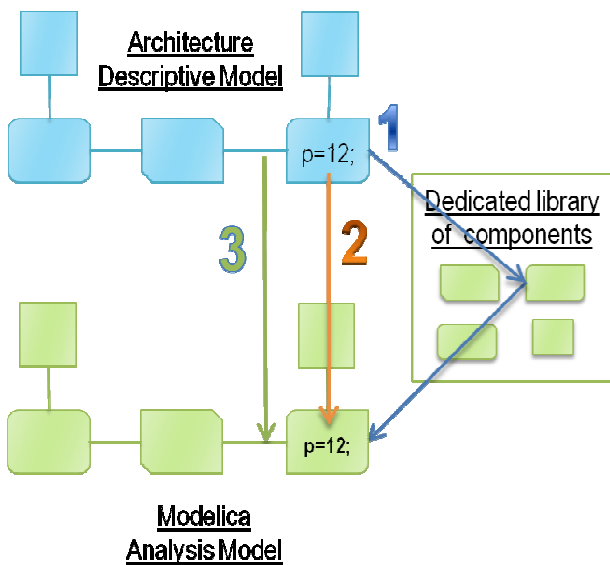


Figure 8 - The three steps of the model Transformations to get the Modelica analysis models

The Modelica analysis module is used in our use case to support sizing analyses and airbrake position control synthesis. As illustrated in the Figure 8, dedicated libraries for these two activities have been used. They will be presented in the next subsection. Then, a model transformation has been developed for each analysis type. Thus, from the same system architecture descriptive model different analyses models can be obtained as illustrated in the figure 9.

This principle is called analysis-based model transformation.

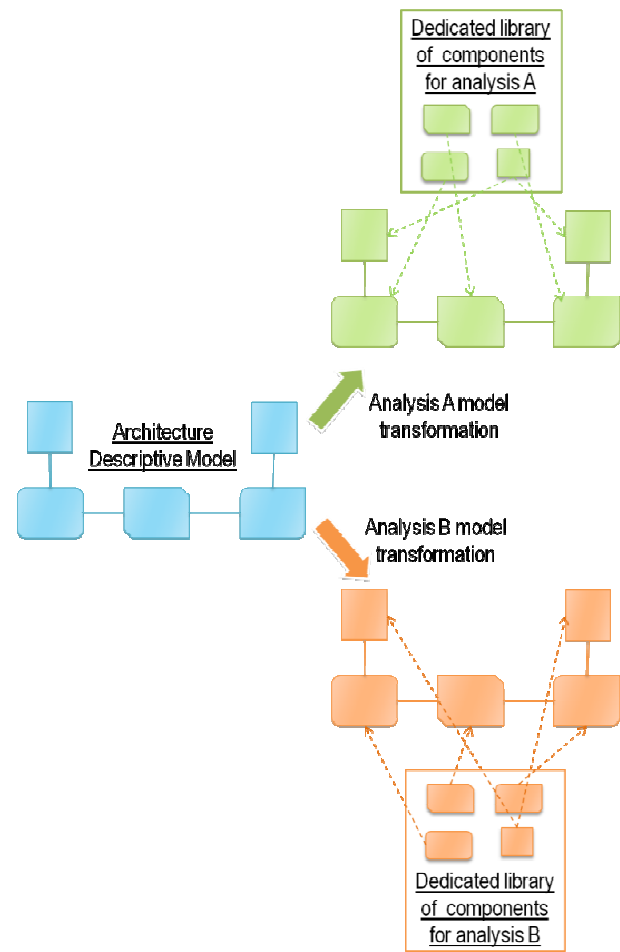


Figure 9 - Analysis-Based Model Transformations

3.6 Analysis-specific libraries

An in-house Modelica library for the preliminary design of electromechanical actuators is being developed since three years in the ICA laboratory of Toulouse [12], [13], [14], [15], [16]. This library uses non-causal models and inverse simulation. In fact, these Modelica models contain estimation, simulation and analysis models. Calculations of different parameters of simulation, sizing and comparison in function of definition parameters are separated in different sub-models. Moreover, according to the type of rated study, models own the necessary and sufficient characteristic sizes for analyze, in order to reduce the parameter number. So as to adapt itself to the various stages of the conception, each physical component has different analysis models according the analysis to be performed. The next figure shows the three analysis models of a component example, the spur gear.

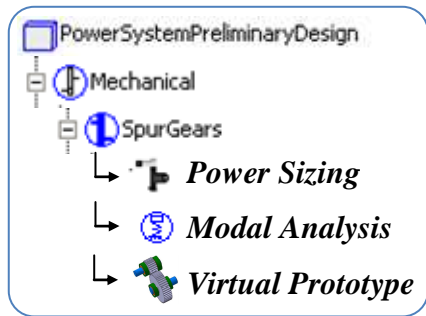


Figure 10 - Example of the three analysis models of the spur gear component

3.6.1 Power Sizing analysis

For the electromechanical actuating system sizing, each component is simulated and estimated with respect to the transmitted power in a backwards way starting from the load and its mission profile (effort and position time histories). This library takes into consideration operating areas, fatigue and reliability of components. Scaling laws approaches are implemented as a fast and efficient strategy in order to reduce the number of design parameters from the numerous model parameters [12]. These models require only inertia and efficiency as details.

3.6.2 Modal analysis for dynamic performances

For the control synthesis, the stability and the speed of the system can be studied with direct simulation. In this preliminary design stage, modal analysis gives the dynamic performance of the structure (pass-band and time response). Thus, more or less complex components models with or without linear friction and linear stiffness, allow the validation of components choice of the EMA..

3.6.3 Validation trough virtual prototype

In a preliminary design phase, components selection is finished and with the virtual prototyping, we start to go up in the V-cycle with more and more detailed models. CAD models of component allow the assembly of EMA elements and the analysis of complex components like carter. Non linearity is integrated in stiffness and in friction to introduce backlash and finer models. Finally, integrating components with complex characteristics, control system and 3D representation, a virtual prototyping is realized to analyse the virtual integration and to go far in synthesis of the global system. This kind of model is realized for a reducer box in the reference [17] where fine phenomena are modelled to implement a virtual prototype.

4 Design scenario and simulation results

A study was performed on the airbrake actuator use case. This study includes the following steps:

1. System architecture definition;
2. Power sizing analysis;
3. Modal analysis;
4. FMECA analysis (not presented).

The next two sections illustrate the analysis performed with Modelica.

4.1 Sizing analysis

The dimensions and mass estimation of one of architecture of the airbrake actuator are summarized in Table 1. Sizing of the mechanical parts and verification of the fatigue constraints was realized from the mission profile. For more precision in methodology refer to the Reference [15].

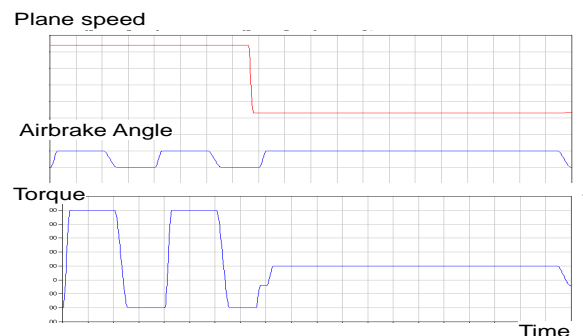


Figure 11: Mechanical mission profile

	<i>Mass (kg)</i>	<i>Length (mm)</i> <i>Diameter</i>
BLDC Cylindrical Motor	1.5	136 67
Brake	0.54	28 73
Spur Gear (ratio=5) + Ball Bearings	0.8	21 129
Roller Screw (pitch=4mm) + Thrust Bearing	1.38	172 53
Rod (hollow)	0.55	113 61
Spherical bearing (2 pieces)	0.14	70 35
Housing (Aluminium)	0.4	252 130
TOTAL MASS	5.3	
FINAL DIMENSIONS		
Distance between hard points		312
Outer diameter		129

Table 1: Components dimensions results

Indeed, components known in power view have modal analysis characteristics associated and a preliminary dynamic study can be realized. In direct simulation, with appropriated models of components, the system modal analysis can be done.

4.2 Modal analysis

For the airbrake actuator, specification imposes stability and time response. So, in a first time, the actuator slaving loop is simplified and composed to proportional controllers for speed slaving and position slaving, with constant disturbance which involves static deviation. Furthermore, the objective of this stage is analyzing natural performance of components in function of dynamic constraint applied.

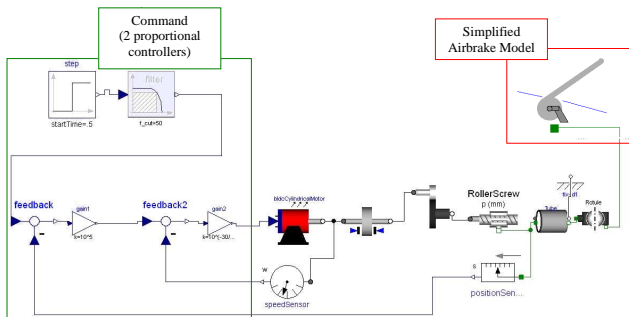


Figure 12 : Modelica slaving model actuator

First, the kinematic of the airbrake model was simplified changing 3D model by inertia and lever arm where aerodynamics load can be applied as a constant load. Secondly, the scaling laws were used to reduce parameter number [12], and assumptions were used to develop more or less complex electro-mechanical models:

- Motor: second order transfer function (cutoff frequency of 500Hz, damping coefficient of 0,7: typical of continuous current motor), torque source and inertia;
- Brake: only inertia (stiffness and backlash unknown);
- Spur Gear: inertia and transmission ratio;
- Roller/Screw: inertia, translation mass, pitch and take into account stiffness or not;
- Rod and Spherical bearing: translation mass and stiffness which can be take into account or not. Thus, anchorage stiffness can be included directly in the spherical bearing model.

The first stage consists in controllers' setting with frequency analysis in open loop. Thus, choice of controllers gain can be done in function of cutoff frequency of the system. Controllers can be adjusted

after some iterations, the first on the speed loop, and then on the position loop.

Then, the answer to a step in input is studied in order to analyze the time response. The output and the input of the slaving are the position of the rod actuator. Of course, the static deviance is still present but the system is stable and time response is less than 1,5 seconds as specification requires (Figure 13).

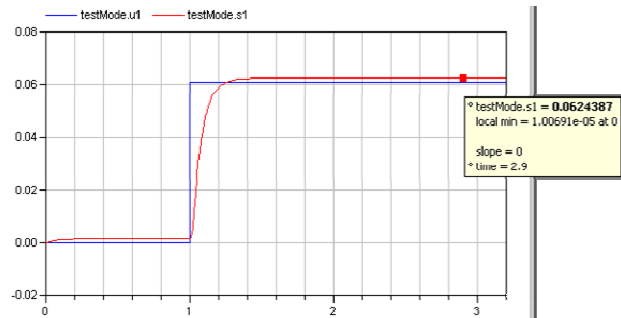


Figure 13: Position system answer to a step

5 Limitations

5.1 Modelica

For system design, Modelica is a very interesting modeling language because of its efficient physical modeling paradigm promoting:

- Object orientation, which allows the re-use of components in different projects with integration of elementary blocs.
- A-causality, thus different analyses are possible such as inverse simulation for power sizing and direct simulation for command synthesis.

However, a designer needs other tools and models to totally design and analyze system architecture, in particular in pre and post processing phases of the simulation:

- Estimation models, prerequisite in the simulation, inform the different necessary parameters for design. These models are static and algebraic. They are implemented in the previously presented library, with scaling laws in definition of some parameters [14] (power laws which reduce the number of definition parameters). For example, the addition of resolution's capacity of algebraic problems, as [18], would allow spreading the range of the treated problems of design.
- Post processing analysis models, which allow processing of results stemming from simulation, in particular in design way to validate components

choice. They are implemented for now, in the in-house library models, with means and integral calculation during the simulation that is weighing it down. It would be better if they could be realized at the end of dynamic simulation.

As an example, for the sizing of components of the system architecture performed in the previous use case the following models are needed. The different models of the spur gear component are separated and described in Modelica.

These aspects of Pre-processing and Post-Processing have to keep the oriented object logic of Modelica language to allow re-using of models. A solution could be the addition in Modelica norms of sections as *static model*, to develop calculation before issuing a dynamic simulation; and *post-processing*, to lead calculation after dynamic simulation; seemed to *equation* form on Modelica language.

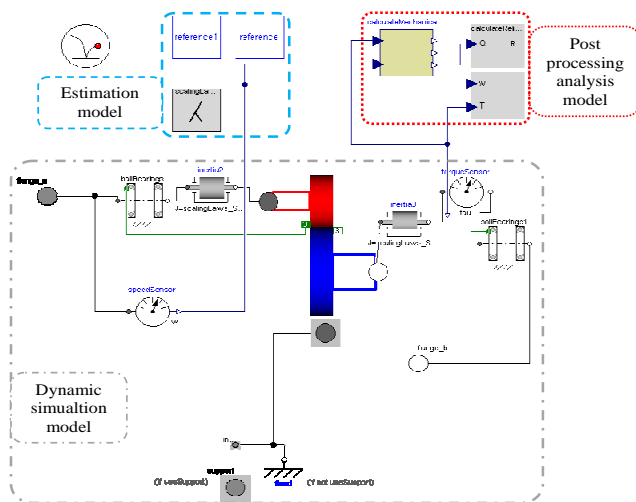


Figure 14: Spur Gear models

5.2 ModelicaML

Regarding the capabilities of the developed prototype, ModelicaML has some limitations:

- ModelicaML doesn't include yet a full simulation center, with an integrated GUI for launching of code generation, compilation, execution and displaying of simulation results on plots inside Eclipse. This means that the Modelica code generated in ModelicaML should at the moment be loaded inside a Modelica simulator outside the prototype. For this use case, Dymola has been used.
- ModelicaML does not allow the import of external Modelica code. This is a real problem to

import and use existing Modelica libraries inside ModelicaML. For this use case we modelled directly in the ModelicaML graphical modelling language the libraries that were necessary.

However these limitations are planned to be removed in a near future by the ModelicaML developers.

6 Conclusions

The main principles of a generic methodology for the development of customized, interoperable and model-driven system design frameworks are illustrated in this paper. This methodology encourages the capitalization of engineering domains' knowledge in order to reuse it by promoting the use of analyses-based model transformations and domain specific modelling languages. It relies on a set of interoperable model-driven tools and languages including EMF, GMF, ATL, or Obeo Designer.

To illustrate this methodology, a domain-specific framework for electromechanical system design was developed. The intended goal of this framework is to be used in early design phases in order to size physical architectures of electromechanical airbrake system. This framework uses the ModelicaML UML profile to support system architecture analyses with the Modelica modelling language. Model transformations from system architecture models to Modelica analysis models are performed through analysis-based model transformations. To this end, we used specific libraries dedicated to preliminary sizing and control/command of electromechanical system. However, the framework doesn't depend only on Modelica for system analysis. As an example document generation capability is implemented in Acceleo for Failure Mode, Effects, and Criticality Analyses. This documentation is not described in this paper but it represents a very important information source for designer as soon the start of design system. The developed framework is just a prototype and should be extended according the methodology principles with other architectural and analyses capabilities.

References

- [1] Doran, T., IEEE 1220: for practical systems engineering. IEEE Computer, Vol.39, No. 5, May 2006.

- [2] Eclipse Modelling Framework Project, <http://www.eclipse.org/modelling/emf/>
- [3] OMG: Meta Object Facility (MOF) 2.0 Core Specification, OMG Document formal/2006-01-01. (2006)
- [4] Jouault, F., Kurtev, I., Transforming Models with ATL. In J.-M. Bruel (Ed.): MoDELS 2005 Workshops, LNCS 3844, p. 128 – 138, 2006
- [5] Acceleo – <http://www.eclipse.org/Acceleo/>
- [6] Juliot, E., Benois, J., La création de points de vue avec Obeo Designer, ou comment fabriquer des DSM Eclipse sans être un développeur expert? In Génie logiciel, GL & IS, p 49 - 54, 2009
- [7] Graphical Modelling Framework., <http://www.eclipse.org/modelling/gmf/>
- [8] Fritzson, P., Principles of Object-Oriented Modelling and Simulation with Modelica 2.1, Wiley-IEEE Press, 2004.
- [9] Chapon, D., Bouchez, G., On the link between Architectural Description Models and Modelica Analyses Models. Proceedings 7th Modelica Conference, Como, Italy, Sep. 20-22, 2009
- [10] Schamai, W., Fritzson, P., Paredis, C., Pop, A., Towards Unified System Modelling and Simulation with ModelicaML: Modelling of Executable Behavior Using Graphical Notations. Proceedings 7th Modelica Conference, Como, Italy, Sep. 20-22, 2009
- [11] OMG: UML OCL 2.0 Specification, OMG Document ptc/2003-10-14. (2003)
- [12] Liscouet, J., "Conception préliminaire des actionneurs électromécaniques - Approche hybride directe/inverse," PhD, Institut Clément Ader, INSA, Université de Toulouse, Chap.3, 2010.
- [13] Budinger, M., Liscouet, J., Lefevre, Y., Fontchastagner, J., Abdelli, A., Allain, L.: Preliminary design of electromechanical actuators with Modelica. Proceedings of the Modelica 2009 Conference (2009)
- [14] Budinger, M., Liscouet, J., Cong, Y., Maré, J.C.: Simulation based design of electromechanical actuators with Modelica. Proceedings of the ASME IDETC/CIE 2009 (2009)
- [15] F. Hospital, M. Budinger, J. Liscouet, J-Ch Maré, "Model Based Methodologies for the Assessment of More Electric Flight Control Actuators", 13th AIAA/ATIO Aviation Technology, Integration and Operation Conference, 13 - 15 Sep 2010 - Fort Worth, Texas.
- [16] M. Budinger, A. Fraj, T. El Halabi, J-Ch. Maré, "Coupling CAD and system simulation framework for the preliminary design of electromechanical actuators", IDMME Virtual Concept, 20-22 October 2010, Bordeaux, France.
- [17] Angelika Peer, Physical-based Friction Identification of an Electro-Mechanical Actuator with Dymola/Modelica and MOPS, Modelica'2003 conference.
- [18] GAMS, <http://www.gams.com/>

Towards a Benchmark Suite for Modelica Compilers: Large Models

Jens Frenkel⁺, Christian Schubert⁺, Günter Kunze⁺, Peter Fritzson^{*}, Martin Sjölund^{*}, Adrian Pop^{*}

⁺Dresden University of Technology, Institute of Mobile Machinery and Processing Machines

^{*}PELAB – Programming Environment Lab, Dept. Computer Science

Linköping University, SE-581 83 Linköping, Sweden

{jens.frenkel, christian.schubert, guenter.kunze}@tu-dresden.de,

{peter.fritzson,martin.sjolund,adrian.pop}@liu.se

Abstract

The paper presents a contribution to a Modelica benchmark suite. Basic ideas for a tool independent benchmark suite based on Python scripting along with models for testing the performance of Modelica compilers regarding large systems of equation are given. The automation of running the benchmark suite is demonstrated followed by a selection of benchmark results to determine the current limits of Modelica tools and how they scale for an increasing number of equations.

Keywords: benchmark, performance comparison, code generation, compiler.

1 Introduction

Benchmarks are a well-known method to compare the capabilities of different software products. Based on the results users are able to choose the best software for their application. Several commercial and non-commercial Modelica compilers are available on the market, like SimulationX, OpenModelica, JModelica, MathModelica, and Dymola.

Due to the growing number of compilers, a tool independent and standardized test is needed from which the strengths of each compiler can be determined. Such a benchmark might also be used by compiler developers to test their compilers for compliance with the Modelica standard. Furthermore it can be used to identify ways of improving simulation performance. This paper tries to develop such a benchmark suite called ModeliMark.

A standard for benchmarking Modelica compilers should cover the following topics:

1. languages features
2. symbolic manipulation power
3. numeric solver robustness
4. compiling performance
5. simulation/target code performance

In the first part all *language features* of Modelica are tested showing the coverage of each compiler.

Symbolic manipulation power refers to testing which simplifications and manipulations are undertaken by the compiler in order to improve simulation speed.

In *Numeric solver robustness* difficult models are simulated comparing their results. Difficult models might feature high indices, inconsistent initial values or singularities which might require dynamic state selection for example [1].

Compiling and simulation performance tests a set of predefined models and measures their time for translation or simulation respectively.

A main concern of this paper is to investigate how current modelica compilers cope with large models, i.e. many equations.

The next chapter gives an overview of previous work on comparisons for Modelica compilers. It is followed by an overview on model design for benchmarking the scalability with respect to model size. Chapter four focuses on how the execution of such a benchmark could be automated using Python. A first glance at some benchmark results is given in the fifth chapter.

2 Previous Work

Every development team of a Modelica compiler already has a wide range of tests to ensure that the compiler is working correctly. Also the Modelica language specification and the Modelica Library include numerous examples which can be included in tests. Some of them can be used to test for *language features* whereas others could be used for performance measurements.

At the Modelica Conference in 2008 [8] a benchmark library focussing on *numerical robustness* was presented. The authors tried to compare their own Modelica compiler MOSILAB with commercial tools. Several models ranging from simple tests for language

features to demanding electrical circuits with discontinuities.

Further possible benchmark models emerged from the efforts of implementing parallel Modelica compilers; see [4] and [5].

2 Large models for Performance Benchmarks

The scope of this paper lies not in trying to establish a general Modelica benchmark but in providing a set of large benchmark models.

With increasing popularity of Modelica the demand for more detailed models increases as well. This leads to larger models with a very high number of equations and variables. However, large systems is a challenging task for Modelica compilers due to the symbolic approach to Modelica compilation.

The following models are supposed to evaluate the performance and boundaries of available Modelica compilers regarding large models. The benchmark comprises a set of synthetic models testing different aspects.

The first model is called *flat model*, containing many variables and equations which are tree structured.

The *hierarchical model* yields similar complexity by recursive use of small submodels.

A further set of models is designed to test the symbolical effort to extract systems of equations. It consists of models with:

- 1 a large number of alias variables, for example “a=b” or “a=-b”, and only a few other equations
- 2 a *linear system* of equations
- 3 a *nonlinear system* of equations
- 4 a linear system of equations with time discrete and continues variables (*mixed linear system*)
- 5 a nonlinear system of equations with time discrete and continues variables (*mixed nonlinear system*)

2.1 Flat Model

The flat model consists of n variables and n equations and has the following form.

```

model flatclass_n
  input Real inp;
  Real v_1;
  Real v_2;
  Real v_3;
  ...
  Real v_n;
equation
  v_1 = 1 + v_2;

```

```

v_2 = 2 + v_3;
...
v_(n-1) = (n-1) + v_n;
der(v_n) = v_1 + inp;
end flatclass_n;

```

The same model may be expressed using a for-loop.

```

model flatClass_N
  constant Integer N=100;
  input Real inp;
  Real v[N];
equation
  for i in 1:N-1 loop
    v[i] = i + v[i+1];
  end for;
  der(v[N]) = v[1] + inp;
end flatClass_N;

```

While both models give the same result their syntax is different leading to different workloads in the compiler. For this comparison only the first model has been considered.

Note that a Modelica compiler which is able to work with for-loops directly instead of expanding them may achieve significantly better results using the second formulation.

2.2 Hierarchical Models

The following hierarchical model features a mechanical system consisting of a long series of masses interconnected by springs. The base class uses the Modelica.Mechanics.Translational library and is shown in Figure 1:



Figure 1. SpringMass Model 1.

In the next level two of these submodels are combined as shown in Figure 2. This step can be repeated for each individual level.

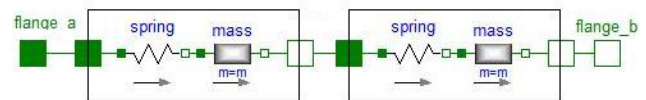


Figure 2. SpringMass Model 2.

The topmost model connects a submodel to a fixed flange. Therefore the number of equations increases with the level as given in Table 1.

Due to the structural information translation may be faster than the flat model with equal number of equations.

Level	Equations
1	21
2	42
3	84
4	168

Table 1: Number of equations for the SpringMass models.

2.3 Alias Model

Connect equations in Modelica models often lead to equations like “a=b” or “a=-b”. Such equations are considered to be alias equations since they merely introduce new names for known variables. Hence, additional alias equations should only lead to minimal overhead.

In order to test this behaviour the model FlatAliasClass has been designed. It is very similar to the FlatModel only with additional alias equations. The number *m* of additional alias equations can be altered to see their influence.

```

model Flataliasclass_n
  input Real inp;
  Real v_1;
  Real v_2;
  ...
  Real v_n;

  Real va_1;
  Real va_2;
  ...
  Real va_m;
equation
  va_1 = v_1;
  va_1 = 1 + v_2;
  va_2 = v_2;
  va_2 = 2 + v_3;
  ...
  va_(n-1) = v_(n-1);
  va_(n-1) = (n-1) + v_n;
  der(v_n) = v_1 + inp;
end Flataliasclass_n;

```

In addition another model called AliasClass_N has been implemented in which alias relations emerge only if previous found aliases are replaced. In fact, if all alias relations are found only the first equation remains.

```

model AliasClass_N
  input Real inp;
  constant Integer N=4;
  Real a[2*N+1];

```

```

equation
  der(a[1]) = inp;
  a[2] = -a[1];
  a[3] = 2*a[2]+a[1];
  for i in 4:2:2*N loop
    a[i] = a[i-3] + a[i-2] - a[i-1];
    a[i+1] = i*a[i]+(i-1)*a[i-1];
  end for;
end AliasClass_N;

```

2.4 Model with linear or nonlinear Systems of Equations

Dealing with linear or nonlinear systems of equations is a basic requirement of every Modelica compiler. The main concern of the next four models is to answer the question up to which size an equation system can be handled by a compiler and what effort it takes.

2.4.1 Linear Model

First, there is the Linersysclass_n which possess a strong connected linear system of *n* equations which has the unique solution:

$$v = 1 + \text{inp}/(n-2) * [1, 1, \dots, 1] \text{ for } n > 2.$$

```

model Linersysclass_n
  input Real inp;
  Real v_0;
  Real v_1;
  Real v_2;
  ...
  Real v_n;
equation
  - v_0 + v_1 + v_2 ... + v_n = n-2 + inp;
  + v_0 - v_1 + v_2 ... + v_n = n-2 + inp;
  + v_0 + v_1 - v_2 ... + v_n = n-2 + inp;
  ...
  + v_0 + v_1 + v_2 ... - v_n = n-2 + inp;
end Linersysclass_n;

```

2.4.2 Mixed Linear Model

Modelica models often include if-equations which lead to time discrete components which themselves may be part of an equation system. Such systems of equations have to be solved using iterative methods which are tested by the following model. Mixedlinersysclass leads to a strongly connected linear system including if-equations.

```

model Mixedlinersysclass_n
  input Real inp;
  Real v_0;
  Boolean b_0;
  Real v_1;
  Boolean b_1;
  Real v_2;
  Boolean b_2;
  ...
  Real v_n;
  Boolean b_n;

```

```

equation
  b_0 = v_0 > 0;
  (if b_0 then -v_0 else -2*v_0) + v_1 +
v_2 ... + v_n = n-2+inp;

  b_1 = v_1 > 0;
  + v_0 + (if b_1 then -v_1 else -2*v_1) +
v_2 ... + v_n = n-2+inp;

  b_2 = v_2 > 0;
  + v_0 + v_1 +(if b_2 then -v_2 else -
2*v_2) ... + v_n = n-2+inp;
  ...

  b_n = v_n > 0;
  + v_0 + v_1 + v_2 ... +(if b_n then -v_n
else -2*v_n) = n-2+inp;
end Mixedlinearsysclass_n;

```

2.4.3 Nonlinear Model

Similar to the linear case, there are models which test the Modelica compiler's ability to solve nonlinear equations by a slight alteration of the aforementioned models.

```

model Nnlinearsysclass_n
  input Real inp;
  Real v_0;
  Real v_1;
  Real v_2;
  ...
  Real v_n;
equation
  - sin(v_0) + v_1 + v_2 ... + v_n=n-2+inp;
  + v_0 - sin(v_1) + v_2 ... + v_n=n-2+inp;
  + v_0 + v_1 - sin(v_2) ... + v_n=n-2+inp;
  ...
  + v_0 + v_1 + v_2 ... - sin(v_n)=n-2+inp;
end Nonlinearsysclass_n;

```

```

model Mixednonlinearsysclass_n
  input Real inp;
  Real v_0;
  Boolean b_0;
  Real v_1;
  Boolean b_1;
  Real v_2;
  Boolean b_2;
  ...
  Real v_n;
  Boolean b_n;
equation
  b_0 = v_0 > 0;
  (if b_0 then sin(v_0) else cos(v_0)) +
v_1 + v_2 ... + v_n = n-2+inp;
  b_1 = v_1 > 0;
  + v_0 + (if b_1 then sin(v_1) else
cos(v_1)) + v_2 ... + v_n = n-2+inp;
  b_2 = v_2 > 0;
  + v_0 + v_1 +(if b_2 then sin(v_2) else
cos(v_2)) ... + v_n = n-2+inp;
  ...
  b_n = v_n > 0;
  + v_0 + v_1 + v_2 ... +(if b_n then
sin(v_n) else cos(v_n)) = n-2+inp;
end Mixednonlinearsysclass_n;

```

3 Automating the Benchmark Suite

The main concern of this paper was to get an answer on how current Modelica compilers cope with large models, i.e. many equations. To get this answer a lot of models with an increasing number of equations had to be translated and simulated. Hence, the generation of the models as well as the control of the Modelica compilers should be fully automated.

The programming language Python proves to be well suited as it allows importing C-Code, starting external processes or even accessing COM-Components under Microsoft Windows. In addition Python is an object oriented script language which is easy to read and for which comprehensive libraries are available.

The first part of the solution is a model generator as shown in Figure 3. It chooses appropriate models, values for n (number of equations) and writes Modelica code which shall then be tested. Each test model is stored in a separate Python class which returns Modelica code for a given n.

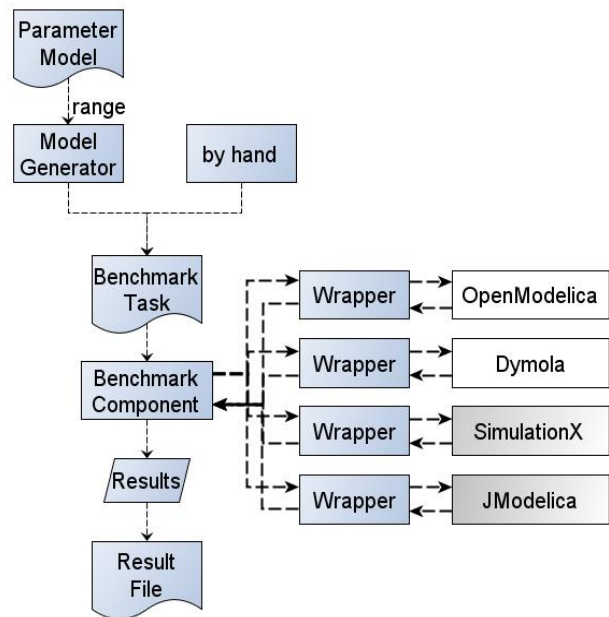


Figure 3. Benchmark Framework.

These models are handed over to the Benchmark Component. It controls the benchmark process and communicates via a wrapper with the corresponding Modelica compiler. Such a wrapper has to be created for each individual compiler. So far wrappers for OpenModelica, JModelica and Dymola have been implemented.

Based on the usual Modelica translation process, which is divided into flattening and symbolic manipulation, the wrappers expose three functions:

- flatten
- translate

- simulate

Flatten instructs the compiler to parse the Modelica code and return the flat model. Translate first flattens the model and turns it into a state which can be simulated (executable for example). Simulate flattens, translates and simulates the model using the standard solver and a predefined output interval and stop time.

Since not every Modelica compiler provides functions to measure the execution time of the flattening, translation and simulation process the functionalities from the Python library *time* is used. All results are written to a text file. The source code as well as the models is freely available at <http://code.google.com/p/modelimark>, and linked from www.openmodelica.org.

4 Benchmark Results

The following benchmarks were accomplished using a Windows 7, 64 Bit System with Intel Core i7 860, 2.80 GHz and 4.0 GB RAM.

4.1 Modelica Compilers

For this benchmark three different Modelica compilers were used:

- OpenModelica compiler Revision 7745 from 21/01/2011
- JModelica 1.4
- Dymola 7.4

4.2 Flat Model

As can be seen in Figure 4 Dymola needs the least time for translation followed by OpenModelica and JModelica.

However time increases roughly with the third power of n which makes Modelica uneconomical for very large models. It was found that the upper limit for the number of equations is not defined by time but by the compiler itself.

While JModelica and OpenModelica failed at around 2000 and 60 000 equations respectively, Dymola managed to translate a model with 160 000 equations but Visual Studio 2008 failed to compile the executable.

4.3 Hierarchical Models

Figure 5 shows the results for the hierarchical Model in comparison to the flat one. It can be seen that JModelica and OpenModelica do not benefit. In Dymola however, the time now only increases with the second power of n . It is assumed that the internal look up process in Dymola exploits the model structure. Nevertheless,

twice the equations still leads to a fourfold time for translation.

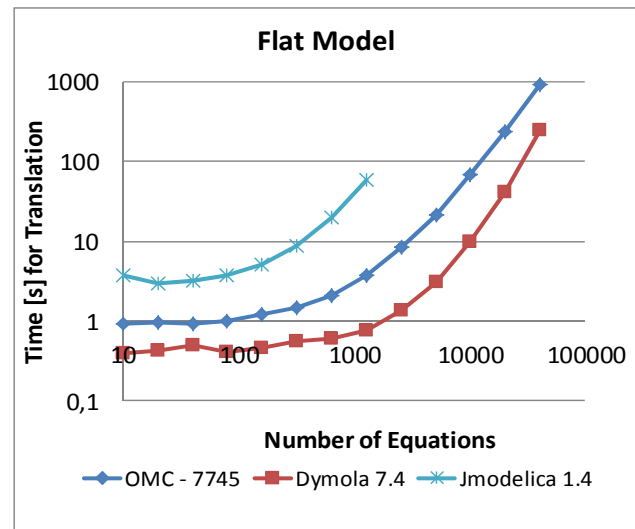


Figure 4: Benchmark Results Flat Model

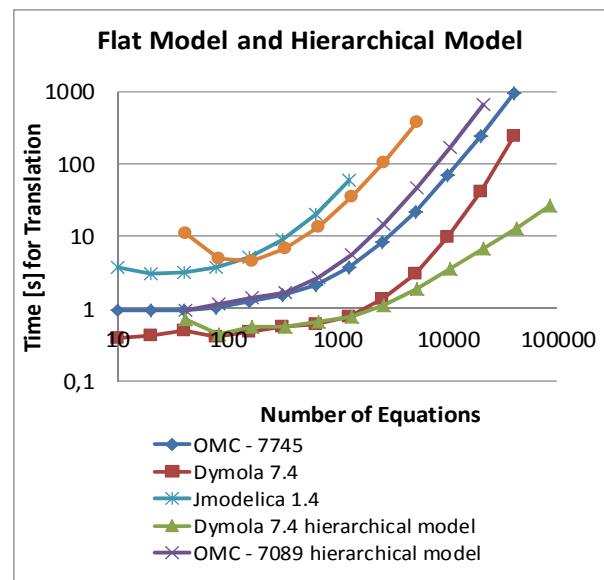


Figure 5: Flat Model and hierarchical Model

4.4 Alias Model

Each graph in the Figures 6, 7, 8 show for each compiler how the time for translation changes with increasing percentage of alias equations for a given number of equations.

In the case of Dymola the influence of alias equations is similar to normal equations.

In OpenModelica and JModelica alias equations are treated more efficiently since their influence is almost linear and independent of n .

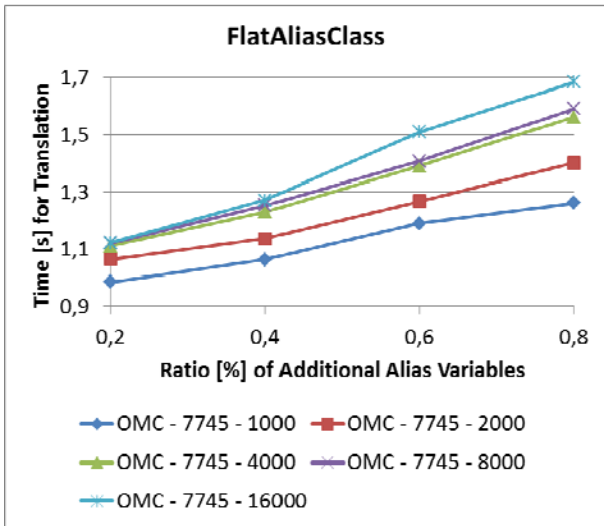


Figure 6: FlatAliasClass - OMC 7745

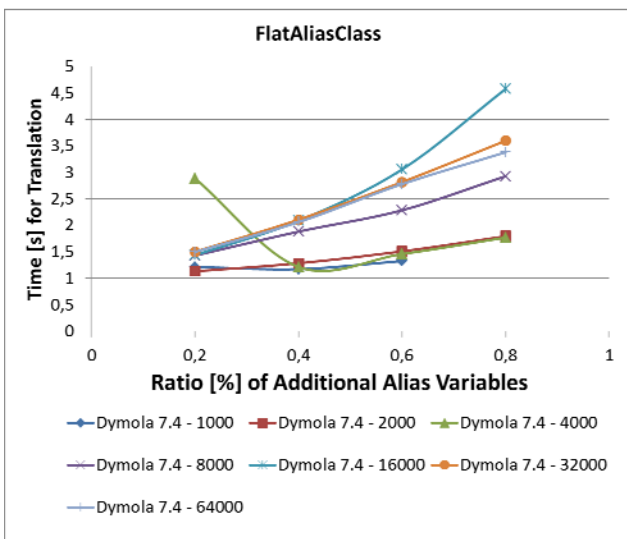


Figure 7: FlatAliasClass Dymola 7.4

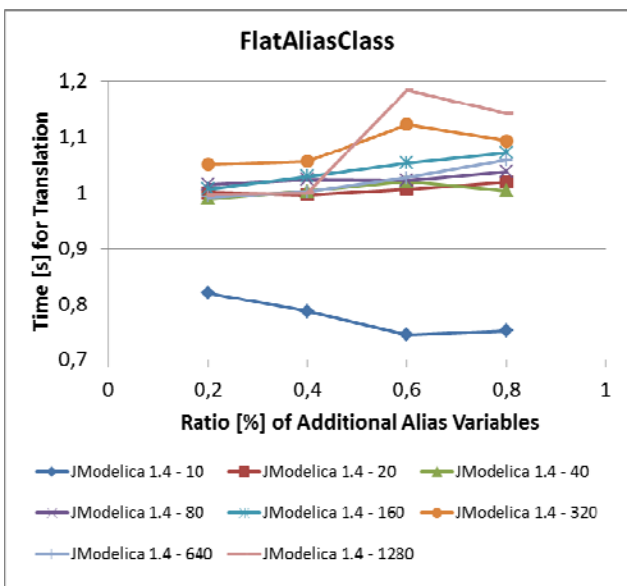


Figure 8: FlatAliasClass JModelica 1.4

For the model `AliasClass` where a recursive substitution is needed, to find all alias relations Dymola seems to be more efficient (Figure 9). Further investigations have shown that the Dymola compiler replaces only the first 11 alias variables. All the other alias variables are not detected and calculated for each simulation step.

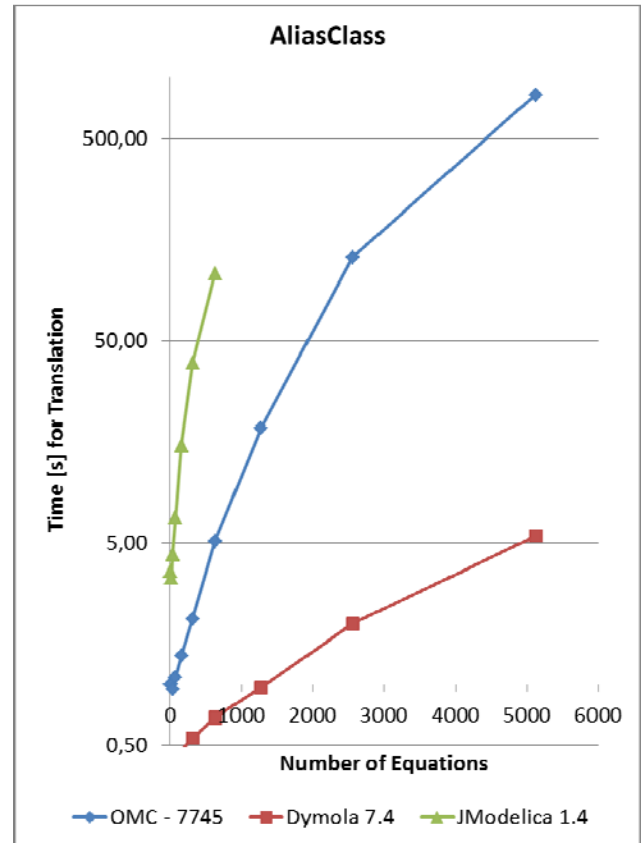


Figure 9: AliasClass Model

4.5 Linear or Nonlinear Systems of Equations

Looking at Figure 10, 11, 12 it can be seen that the results for the different types of equation systems hardly differ. Again, the implementation in Dymola seems to be more efficient compared to JModelica and OpenModelica. Note, that the maximum number of unknowns which could be solved for in Dymola was 320 compared to a 160 in OpenModelica and 80 in JModelica.

5 Conclusions

This paper tried to encourage the development of a standard benchmark suite for Modelica compilers. It would give compiler developers insights to find possibilities for improvements and give users the chance to compare different compilers.

The scope of this paper was limited to the behavior of current Modelica compilers regarding large models. It could be found that Dymola was generally faster than OpenModelica and JModelica. However, even Dymola does not seem suitable for very large models as it cannot cope with models that have more than 160000 equations.

Furthermore it was found that, depending on the model, the time needed for translation grows with second or third power of the number of equations.

In order to continue establishing Modelica as the major simulation language better ways of dealing with large models have to be found. Some first promising ideas are given in [6] and [7].

All the models as well as the Python code are freely available at <http://code.google.com/p/modelimark/>, and linked from www.openmodelica.org.

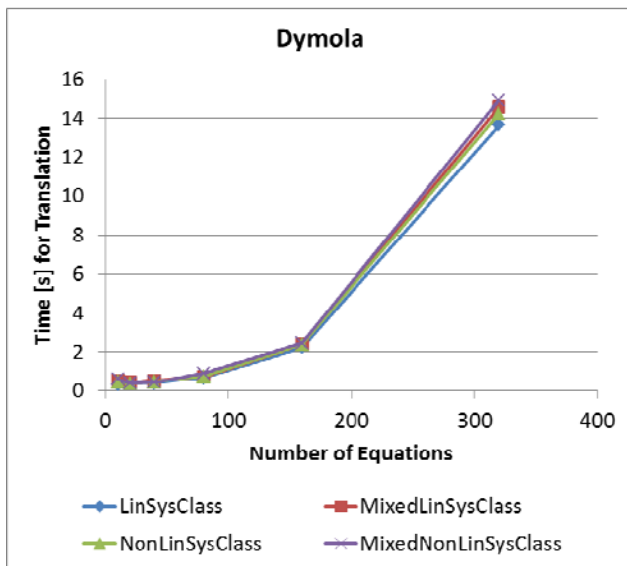


Figure 10: Linear and Nonlinear Systems of Equations Dymola 7.4

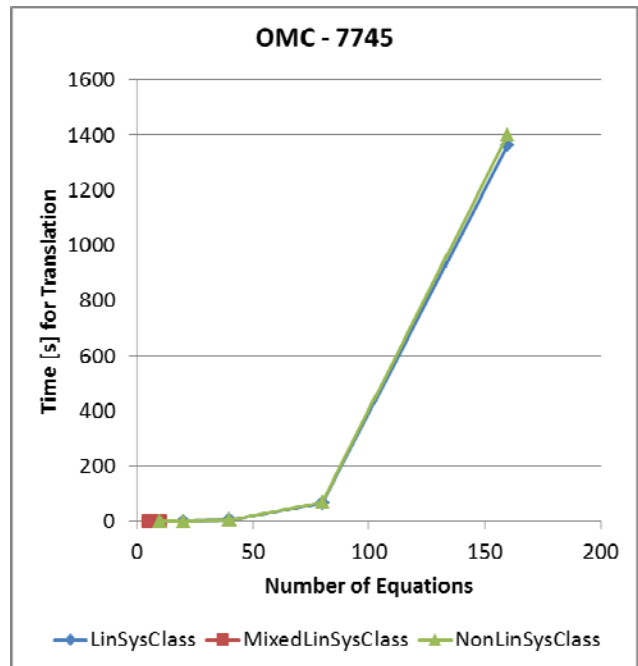


Figure 11: Linear and Nonlinear Systems of Equations OMC - 7745

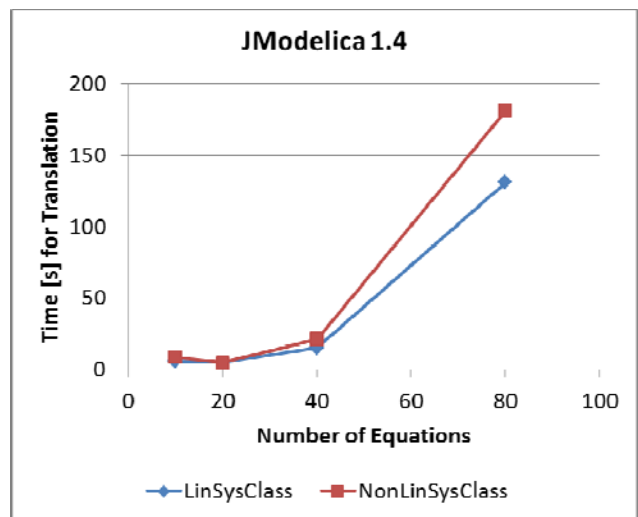


Figure 12: Linear and Nonlinear Systems of Equations JModelica 1.4

References

- [1] S. Mattsson, G. Söderlind: Index reduction in differential-algebraic equations using dummy derivatives. *SIAM J. Sci. Statist. Comput.*, 14:677– 692, 1993.
- [2] Peter Fritzson: *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*, Page 57ff, Wiley IEEE Press, 2004.
- [3] O. Enge-Rosenblatt, C. Clauß, P. Schwarz, F. Breitenecker, C. Nytsch-Geusen: Comparison of Different Modelica-Based Simulators Using Benchmark Tasks, in *Proceedings of Modelica Conference 2008*
- [4] M. Maggio, K. Stavåker, F. Donida, F. Casella, P.Fritzson: Parallel Simulation of Equation-based Object-Oriented Models with Quantized State Systems on a GPU, in *Proceedings of the 7th Modelica Conference 2009*
- [5] H. Lundvall, K. Stavåker, P. Fritzson, C. Kessler: Automatic parallelization of simulation code for equation-based models with software pipelining and measurements on three platforms. in *ACM SIGARCH Computer Architecture News*, Vol. 36, No. 5, December 2008.
- [6] D Zimmer: Module-Preserving Compilation of Modelica Models, *Proceedings 7th Modelica Conference, Como, Italy, Sep. 20-22, 2009.*
- [7] C. Höger, F. Lorenzen, P. Pepper: Notes on the Separate Compilation of Modelica, *The 3rd International Work-shop on Equation-Based Object-Oriented Modeling Languages and Tools*, Oslo, Norway, October 3, 2010

Appendix

a. FlatModel

n	OMC - 7745		Dymola 7.4		Jmodelica 1.4	
	T	S	T	S	T	S
10	0,92	0,11	0,39	0,11	3,73	0,21
20	0,94	0,07	0,42	0,09	3,00	0,15
40	0,91	0,09	0,50	0,03	3,19	0,24
80	1,00	0,13	0,41	0,19	3,73	0,25
160	1,21	0,21	0,46	0,14	5,04	0,50
320	1,49	0,35	0,55	0,33	8,73	1,45
640	2,11	0,68	0,59	0,55	19,76	5,66
1280	3,73	1,27	0,75	1,11	59,57	22,30

2560	8,37	2,61	1,34	2,00		
5120	21,75	5,01	3,05	4,06		
10240	69,13	10,01	9,83	7,66		
20480	239,78	21,47	41,18	16,25		
40960	932,63	42,87	245,29	35,01		

b. Hierarchical Models

n	OMC - 7745		Dymola 7.4		JModelica 1.4	
	T	S	T	S	T	S
42	0,94	0,21	0,70	0,27	10,72	1,57
84	1,14	0,13	0,44	0,07	4,86	0,15
168	1,39	0,3	0,55	0,00	4,55	0,23
336	1,61	0,29	0,56	0,11	6,78	0,41
672	2,63	0,57	0,66	0,13	13,52	1,13
1344	5,58	1,07	0,76	0,35	34,38	3,59
2688	14,67	2,14	1,10	0,36	103,74	13,30
5376	46,60	4,47	1,86	1,70	377,37	50,66
10752	165,47	10,26	3,58	0,94		
21504	666,54	26,01	6,69	7,21		
43008			13,03	7,29		
86016			26,37	46,29		

c. FlatAliasClass

n	alias	OMC - 7745		Dymola 7.4	
		T	S	T	S
1000	0	3,27	1,57	0,59	0,25
1200	200	3,22	1,83	0,72	0,82
1400	400	3,47	2,01	0,69	1,06
1600	600	3,89	2,15	0,79	1,13
1800	800	4,13	1,91	0,79	1,27
2000	0	6,52	2,24	1,03	1,65
2400	400	6,94	2,68	1,16	1,77
2800	800	7,41	2,93	1,32	2,06
3200	1200	8,26	3,67	1,55	2,30
3600	1600	9,14	3,92	1,85	2,41
4000	0	15,80	4,13	2,27	3,10
4800	800	17,56	5,02	2,89	3,34
5600	1600	19,46	5,90	3,50	3,92

6400	2400	21,99	6,61	4,20	4,52
7200	3200	24,69	7,53	5,10	5,19
8000	0	47,53	8,19	6,48	5,97
9600	1600	53,25	9,96	9,25	6,91
11200	3200	59,60	11,45	12,18	7,71
12800	4800	66,90	12,48	14,79	9,23
14400	6400	75,67	14,36	18,97	9,43
16000	0	155,14	16,84	24,88	11,30
19300	3200	174,05	20,12	35,72	14,21
22400	6400	197,41	23,53	51,89	15,18
25600	9600	234,25	26,64	76,05	16,55
28800	12800	261,50	29,90	113,93	9,23
32000	0			114,03	23,75
38400	6400			216,26	28,89
44800	12800			303,47	35,67
51200	19200			405,83	32,61
57600	25600			518,32	32,39
64000	0			669,37	39,99
76800	12800			1001,91	107,16
89600	25600			1379,49	160,05
102400	38400			1860,19	44,43
115200	51200			2262,47	33,65

144	64	3,63	0,23
160	0	4,87	0,48
192	32	4,91	0,47
224	64	5,02	0,49
256	96	5,13	0,47
288	128	5,22	0,51
320	0	8,29	1,38
384	64	8,70	1,38
448	128	8,75	1,40
512	192	9,30	1,43
576	256	9,06	1,42
640	0	19,37	5,37
768	128	19,22	5,27
896	256	19,40	5,21
1024	384	19,89	5,28
1152	512	20,50	5,31
1280	0	60,08	21,35
1536	256	59,92	21,11
1792	512	60,04	21,20
2048	768	71,17	21,93
2304	1024	68,60	22,68

n	alias	JModelica 1.4	
		T	S
10	0	3,64	0,18
12	2	2,99	0,21
14	4	2,87	0,14
16	6	2,72	0,12
18	8	2,74	0,14
20	0	2,85	0,14
24	4	2,85	0,15
28	8	2,84	0,13
32	12	2,86	0,15
36	16	2,90	0,15
40	0	3,04	0,15
48	8	3,00	0,18
56	16	3,04	0,16
64	24	3,10	0,18
72	32	3,05	0,18
80	0	3,50	0,25
96	16	3,55	0,26
112	32	3,58	0,23
128	48	3,57	0,26

d. AliasClass

n	OMC - 7745		Dymola 7.4		JModelica 1.4	
	T	S	T	S	T	S
10	0,99	0,08	0,04	0,10	3,63	0,29
20	1,00	0,10	0,41	0,15	3,36	0,18
40	0,94	0,14	0,45	0,11	4,39	0,32
80	1,09	0,21	0,44	0,19	6,73	0,61
160	1,39	0,49	0,48	0,31	15,26	2,00
320	2,11	0,66	0,54	0,55	38,80	6,06
640	5,10	1,29	0,69	0,99	107,41	25,03
1280	18,53	2,59	0,96	1,95		
2560	128,74	5,06	2,00	3,71		
5120	820,39	10,45	5,42	7,41		

e. LinSysClass

n	OMC - 7745		Dymola 7.4		JModelica 1.4	
	T	S	T	S	T	S
10	1,05	0,15	0,34	0,08	5,64	0,18
20	1,45	0,20	0,39	0,06	4,70	0,16
40	5,33	0,41	0,42	0,09	15,31	0,25
80	66,56	2,06	0,68	0,09	131,26	0,49
160	1363,40	10,92	2,24	0,12		
320			13,64	0,11		

f. MixedLinSysClass

n	OMC - 7745		Dymola 7.4	
	T	S	T	S
5	1,05	0,06	0,36	0,02
10	1,22	0,08	0,54	0,02
20			0,44	0,06
40			0,54	0,08
80			0,73	0,18
160			2,39	0,26
320			14,55	0,34

g. NonLinSysClass

n	OMC - 7745		Dymola 7.4		JModelica 1.4	
	T	S	T	S	T	S
10	1,03	0,11	0,45	0,02	8,76	1,24
20	1,43	0,25	0,38	0,06	4,70	0,14
40	5,37	1,02	0,47	0,09	21,50	0,67
80	69,56	6,38	0,76	0,11	181,44	0,42
160	1402,46	45,11	2,36	0,24		
320			14,24	0,38		

h. MixedNonLinSysClass

n	Dymola 7.4	
	T	S
10	0,60	0,23
20	0,41	0,11
40	0,45	0,09
80	0,90	0,13
160	2,44	0,30
320	14,93	0,61

OMWeb – Virtual Web-based Remote Laboratory for Modelica in Engineering Courses

Mohsen Torabzadeh-Tari, Zoheb Muhammed Hossain, Peter Fritzson, Thomas Richter¹
 PELAB – Programming Environment Lab, Dept. Computer Science
 Linköping University, SE-581 83 Linköping, Sweden
 {mohto, x10muhho, petfr }@ida.liu.se
¹Rechenzentrum, Stuttgart University, Germany
¹richter@rus.uni-stuttgart.de

Abstract

In this paper we present a web-based teaching environment, OMWeb, useful both in engineering courses as well as for teaching programming languages. OMWeb can be an alternative or complementary tool to the traditional teaching method with lecturing and reading textbooks.

Experience shows that using such interactive platforms will lead to more engagement from the students. With such a solution the student can focus more on the important learning goals. The student needs only to open a web browser and start writing programs in order to use the tool. The OMWeb server contains all the needed software. In each interaction, the server returns results to the user. This solution allows each student to work at his/her own speed, at any time, and remotely, enhancing the individual learning.

OMWeb is part of the open source platform OpenModelica. It can be applied to several areas in natural science, such as physics, chemistry, biology, biomechanics etc., where phenomena can be illustrated by dynamic simulations.

Keywords: OMWeb, OpenModelica, Virtual, Web-based

1 Introduction

In this paper we introduce a learning environment for web-based modern object-oriented equation-based modeling and simulation. This environment, called OMWeb, is useful both in programming language teaching and in engineering courses. The primary application shown in this paper is teaching the Modelica language [1]. However, the concept can also be adapted to other languages. In this way the student has an interactive common platform for learning programming languages as well as learning through virtual experiments with physical phenomena.

This kind of interactive course allows experimentation and dynamic simulation as well as execution of computer programs. As a part of the open source platform OpenModelica, [2], this makes it possible to integrate applied sciences in physics, human biology [3], mathematics, and computer science.

2 OpenModelica Platform

In 2002 an initiative was taken by the PELAB group at Linköping University to develop an open source platform for the Modelica language, to be called OpenModelica [2],[3] and [10]. The OpenModelica effort has expanded, and is in recent years also supported by the Open Source Modelica Consortium.

The OpenModelica environment, shown in Fig. 1, consists of several interconnected subsystems. The debugger currently supports debugging of an extended algorithmic subset of Modelica, MetaModelica.

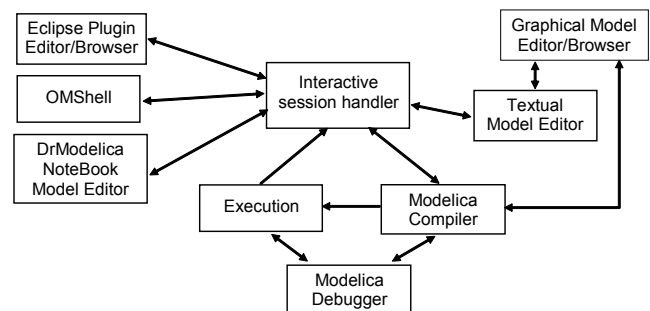


Figure 1. Illustration of communication between different parts of the OpenModelica platform.

The OpenModelica Notebook editor, OMNotebook (see Section 5.3), provides an active electronic notebook including an editor. The notebook is active in the sense that models inside the book can be changed and executed, it is not just a passive textbook or html page. This is one of the first open source efforts that makes it possible to create interactive books for educational purposes in general, and more specifically for teaching and

learning programming. Traditional teaching methods with lecturing and reading a textbook are often too passive and don't engage the student as much.

3 OMWeb architecture

The OMWeb architecture is composed of three decoupled set of entities namely the *Teacher* and/or *Student Client* (TC and SC), the *E-learning Community Server* and the *Computation Client*, similar to the NumLab architecture (Section 5.1).

The three layers have been developed in different programming languages; the clients are developed in Java, the E-learning Community Server (ECS) is developed in Ruby On Rails and for the Computation Client (CC) C++ is used, see Fig 2.

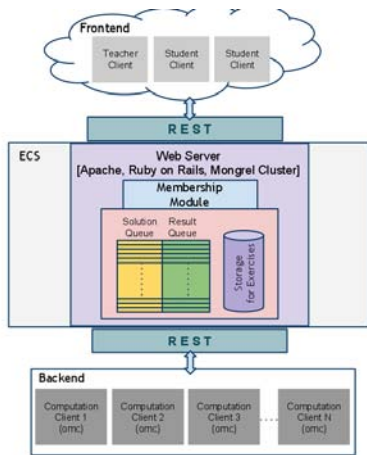


Figure 2. OMWeb architecture

Communication among the modules is done following the REpresentational State Transfer (REST) architecture; which uses Hyper Text Transfer Protocol, HTTP as the carrier of messages across the network. The HTTP has four methods for accessing and updating the resources - GET, POST, PUT and DELETE.

Moreover, JavaScript Object Notation (JSON) is used to format the data for communication. One reason behind the choice of JSON format is that it is easy for the humans to read and write as well as for the machines to parse the data.

4 OMWeb – OpenModelica Virtual Web-based Learning Platform

In this Section the different parts of OMWeb are explained in detail.

OMWeb provides a programming environment within a web browser. This facilitates for the student to learn and participate in courses, independent of time and place. The availability in a standard web browser

makes it easier to get started with than if you have to install special software packages.

The *Student Client* and the *Computation Client* are both active entities in the system, whereas ECS is the passive entity. In detail, the ECS never initiates a communication, rather it only responds to occurring events.

On the other hand the end clients are the ones who are always polling for Solution or Result messages from the Solution and Result Queue of the ECS as soon as they finish POST-ing a message to the ECS. The communication diagram in Fig 6 reflects a better illustration of the message flow in the System.

4.1 Frontend: Teacher Client, TC

The TC [13] is the web frontend of the system and is specially developed for a teacher to post his/hers exercises or assignments intended for the students to solve and get evaluated.

Fig. 3 illustrates the Graphical User Interface of this client. The tab "Exercise Generator " has three fields for entry; the name of the exercise, the description of the exercise and the text area takes in the program code.

The teacher then clicks on the "Send Exercise" button; this action will first generate the JSON script of the corresponding exercise by filling in the "value" tag of the respective "identifier". Later, it sends the JSON string using the HTTP POST which posts the exercise to the ECS.

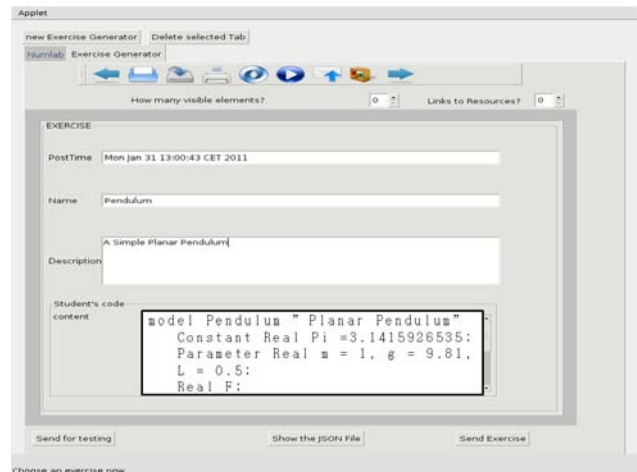


Figure 3. Teacher Client

4.2 Frontend: Student Client, SC

The SC [14] is a web frontend designed for the students who are to solve the exercises posted by the TC, see Fig 4.

A student first opens the web applet and clicks on the drop down box of the client user interface that is

labeled "Fetch Exercise" to retrieve the list of exercises posted by their teacher. The student then selects one of the exercises from the list and the exercise is shown in the client interface.

The exercise may contain one or more sections labeled as Editable and Non-editable where the students can edit the code of the Editable section(s) only.

Later, when the student is done with solving the program, the button labeled "Execute" is pressed, this will first generate the JSON string for the Solution of the exercise and then will send over the network to the ECS using the HTTP POST method.

As soon as sending of the solution is done the SC initiates to poll for the Result of the Solution from the ECS.

The client continues to poll until it receives the Result JSON string from the ECS. On receiving the Result the JSON string is parsed and the result data is shown on the output section of the client interface.

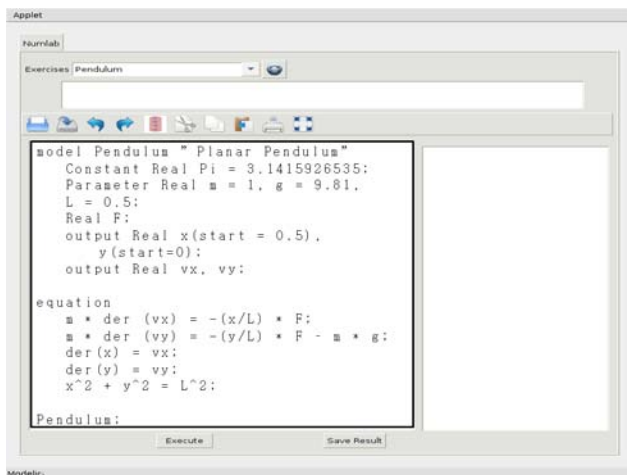


Figure 4. Student Client

4.3 Middleware, E-learning Community Server, ECS

The ECS module works as the middleware between the frontend client and the backend client (server) in order to forward the requests back and forth.

The ECS is composed of three internal modules which are used to manage the messages of the clients: Community, Membership and Resources.

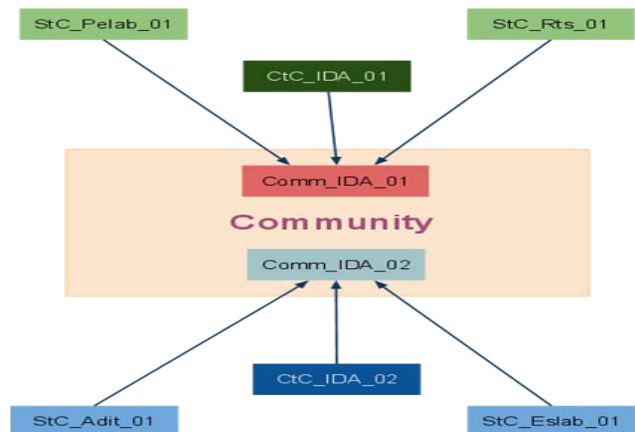


Figure 5. ECS Community

In order for the clients at both ends to communicate with each other they are required to be first registered to the ECS with a unique Membership ID where the membership is granted by the administrator of the ECS.

Next, the Members are assigned to a common Community (Fig. 5), this is required to route the messages of the same community members back and forth.

For example let us assume that there are four frontend SCs and two backend Computation Clients with membership IDs StC_Pelab_01, StC_Rts_01, StC_Adit_01, StC_Eslab_01 and CtC_IDA_01, CtC_IDA_02 respectively. Also assume that there are two communities in the ECS Comm_IDA_01 and Comm_IDA_02.

Let us also assume that client StC_Pelab_01, StC_Rts_01 and CtC_IDA_01 are members of the Comm_IDA_01 community and that clients StC_Adit_01, StC_Eslab_01 and CtC_IDA_02 are members of the Comm_IDA_02 community.

Now, the messages sent by the clients StC_Pelab_01 and StC_Rts_01 can be processed by the backend client CtC_IDA_01 only and messages sent by the StC_Adit_01, StC_Eslab_01 would be processed by the CtC_IDA_02 only.

This kind of routing guarantees that the messages are received by the intended entities only and hence eliminates any misrouting possibilities.

The ECS also maintains a database to store the exercises posted by the teachers using an authentic TC. The exercises are given a unique ID in order to be identified by the front/backend clients while generating the Solutions and the Results, further about this is discussed in later segment.

To manage the messages from and to the clients two separate queues are maintained. These queues are event driven queues supplied by the Ruby On Rails architecture. An event handler takes care of the specific events generated by the incoming messages at the ECS. When

the ECS receives an incoming message from the Student Client containing the HTTP method POST, the event handler routes the message to the Solution Queue and when the ECS receives an incoming message from the Computation Client with the HTTP method POST, it is put to the Result Queue.

Similarly, when the incoming message from the Student Client and Computation Client is a HTTP GET, the messages are retrieved from the Result and Solution Queue respectively and forwarded to the clients who initiated the GET request.

4.4 Backend: Computation Client, CC

The CC is responsible for executing the Solutions sent by the SC, evaluate the correctness of the program code and send the Result back to the ECS.

As we have mentioned in the previous section, the clients who are member of the same community can communicate between themselves; so, the computation client CtC01 could only fetch, execute and evaluate the result that was sent by the Student Client StC01.

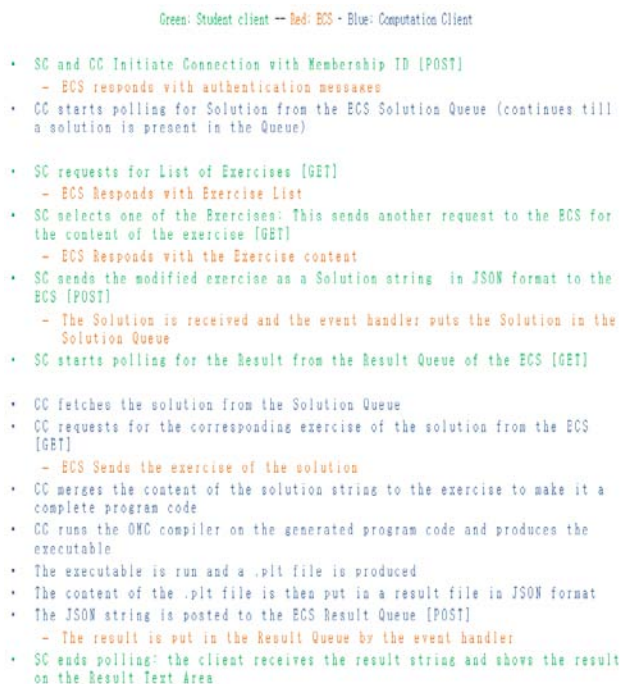


Figure 6. Sequence of message flow

The CC is developed in C++ on a Linux platform. It adopts Sandbox technique to limit the program instance accesses, e.g. only the Linux commands available inside the Sandbox. When the CC fetches a Solution message from the Solution Queue of the ECS it carries out several sequential tasks in order to evaluate the program code. First, it parses the solution JSON string and extracts the respective exercise of the solution from the ECS. Then the core solution content is extracted

from the JSON string and is merged to the "editable" section of the exercise; which makes it a complete program. A file with the program's name is then created and the generated program code is copied and pasted into it.

Next, the type of compiler that should be used to compile the program is extracted from the JSON string with the respective flags and the program file is compiled, in our case it is the Modelica compiler. If there is any error during the compilation, a result JSON string is generated with the compiler error message and is posted to the ECS Result Queue.

Otherwise, on successful compilation a Makefile is produced which is then executed. The execution of the Makefile creates an executable, it is then executed and on successful execution, a .plt file is created.

The content of the .plt file is then pasted in the result JSON string and posted back to the ECS Result Queue, eventually which is polled by the specific Student Client.

5 Related Work

A brief survey is presented in this section covering some existing web-based and interactive learning platforms.

5.1 NumLab Architecture

At University of Stuttgart a web-based virtual environment NumLab [7] is available for computerized mathematical calculations including related subjects. The idea is to provide a web-based teaching environment where the students can focus on the numerical and mathematical topics without having to install any software packages.

NumLab is built according to the client-server architecture with a Java-applet in the front-end, a middleware layer E-learning Community Server, and a back-end client containing all the involved software packages, [9].

For communication between the ECS server and the other parts of the system REST, Representational State Transfer, is used which simplifies the communication in web-based distributed systems. The data representation and exchange format JSON, JavaScript Object Notation is used, [8].

5.2 Intelligent Tutoring System

There are many web-based intelligent tutoring systems that are worth to mention. For example the ELM-ART used for teaching the Lisp language [12] or a plugable web-based tutoring system in [11].

5.3 DrModelica

The OMNotebook subsystem in OpenModelica is currently being used for course material (DrModelica) in teaching the Modelica language and equation-based object-oriented modeling and simulation, (see Fig 7).

It can easily be adapted for use with electronic books teaching other programming languages. OMNotebook can also easily be used in other areas such as physics, biology chemistry, biomechanics etc., where phenomena can be illustrated by dynamic simulations within the book.

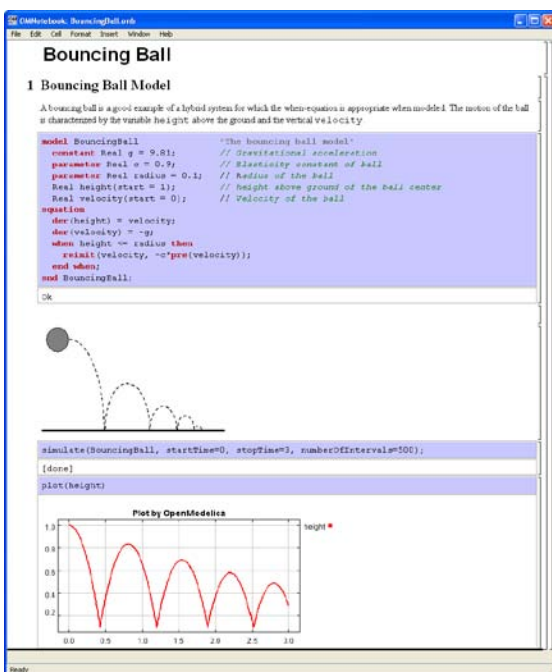


Figure 7. Bouncing ball example with movement animation in OMNotebook

5.4 OMScheme

With OMScheme the OMNotebook paradigm is generalized towards other programming languages than Modelica, e.g the Scheme programming language, [6]. An implementation of the factorial function using OMScheme is shown in Fig 8.

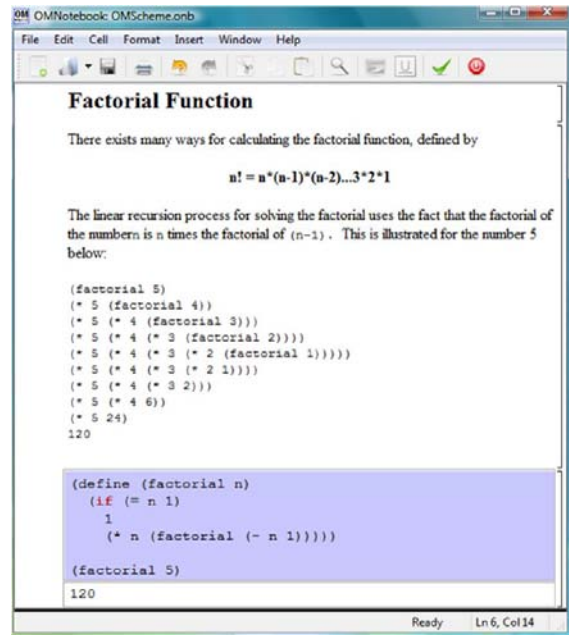


Figure 8. Factorial function illustrated in OMScheme

5.5 DrControl

DrControl, Fig 9, is a recently developed active electronic book course material based on OMNotebook for teaching control theory and modeling with Modelica.

It contains explanations about basic concepts of control theory along with Modelica exercises. Observer models, Kalman filters, and linearization of non-linear problems are some of the topics in the course used in control of a pendulum, a DC motor, and a tank system model among others.

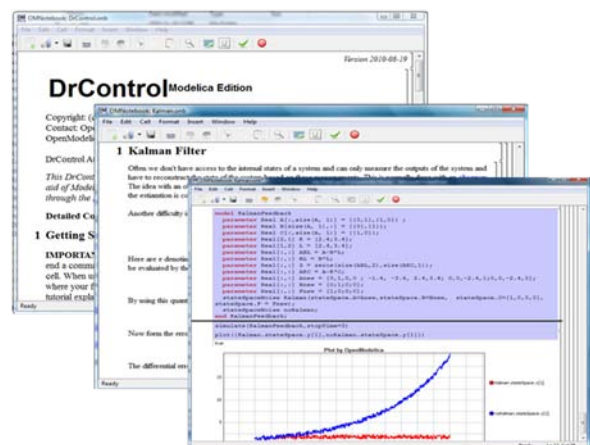


Figure 9. DrControl for teaching control theory concepts.

6 Future Work

The OMWeb platform presented in this paper is pure-text based. Integrating the graphical connection editor, OMEdit into OMWeb would be one of the desired next mile-stone. Also a 3D visualization and syntax highlighting should be supported for making the environment more user friendly.

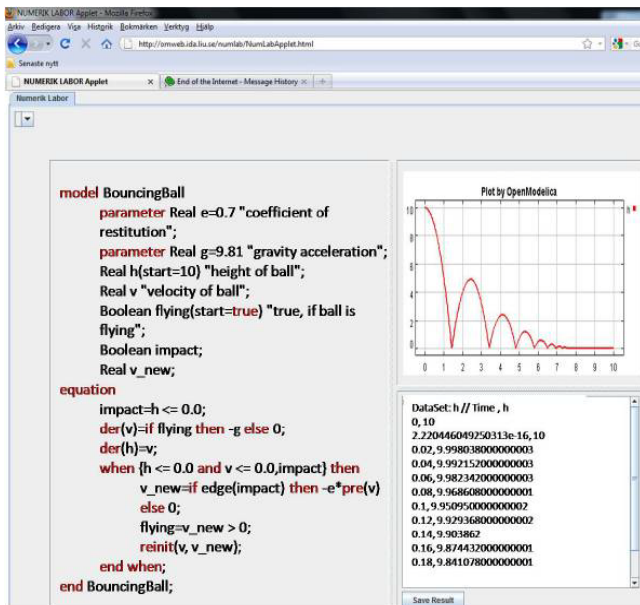


Figure 10. Bouncing Ball illustrated in OMWeb with syntax highlighting aid and plotting.

7 Conclusions

In this work we extended the basic idea of an active web-based teaching environment for educational purposes to handle multiple programming languages.

An early prototype is being developed for handling the Scheme and Modelica languages. OMWeb takes the virtual remote learning environment idea further by introducing OpenModelica platform within NumLab and widens the applicability to a wide range of engineering courses by introducing Modelica language in those courses.

The benefits and opportunities offered by a web-based solution is natural to access a vast amount of knowledge and information but also the ability for the student to work at his or hers own speed which enhances the learning process. Furthermore, the student activity is encouraged more by the interactivity and ease-of-use within an easy-to-use web-based interface

8 Acknowledgements

This work has been supported by EU project Lila and Vinnova in the ITEA2 OPENPROD project. The Open Source Modelica Consortium supports the OpenModelica work.

References

- [1] Modelica Association. The Modelica Language Specification Version 3.1, May 2009. www.modelica.org
- [2] Peter Fritzson et al OpenModelica Users Guide and OpenModelica System Documentation, www.ida.liu.se/projects/OpenModelica, 2009.
- [3] Anders Sandholm, Peter Fritzson, Varun Arora, Scott Delp, Göran Petersson, and Jessica Rose. The Gait E-Book - Development of Effective Participatory Learning using Simulation and Active Electronic Books. In *Proceedings of the 11th Mediterranean Conference on Medical and Biological Engineering and Computing (Medicon'2007)*, Ljubljana, Slovenia, June 26 - 30, 2007.
- [4] Bernhard Bachmann, Peter Aronsson, and Peter Fritzson. "Robust Initialization of Differential Algebraic Equations" In Proc. of (Modelica '06), Vienna, Austria, 2006.
- [5] Mohsen Torabzadeh-Tari, Peter Fritzson, Adrian Pop, and Martin Sjölund. Generalization of an Active Electronic Notebook for Learning Multiple Programming Languages, IEEE EDUCON Education Engineering 2010 – The Future of Global Learning Engineering Education, Madrid, Spain, 2010
- [6] Anders Fernström, Ingemar Axelsson, Peter Fritzson, Anders Sandholm, Adrian Pop. OMNotebook – Interactive WYSIWYG Book Software for Teaching Programming. In *Proc. of the Workshop on Developing Computer Science Education – How Can It Be Done?*. Linköping University, Dept. Computer & Inf. Science, Linköping, Sweden, March 10, 2006
- [7] Bankolé Adjibadji, Stephan Rudolf, and Thomas Richter. Numerische Mathematik im Browser: Das Virtuelle Programmierlabor ViP, Stuttgart University, Rechenzentrum, Feb 2010 <http://isblab.rus.uni-stuttgart.de:7070/numlab/exercises>
- [8] Douglas Crockford. Introducing JSON, URL: <http://json.org>. Retrieved March 25, 2010
- [9] Heiko Bernloehr. E-learning Community Server, URL: <http://freeit.de/ecsa/index.html>, Retrieved March 25, 2010
- [10] Peter Fritzson, Peter Aronsson, Håkan Lundvall, Kaj Nyström, Adrian Pop, Levon Saldamli, and

David Broman. The OpenModelica Modeling, Simulation, and Software Development Environment. In Simulation News Europe, 44/45, December 2005. See also: <http://www.openmodelica.org>.

- [11] Ang Yang, Kinshuk, Ashok Patel, A Plug-able Web-based Intelligent Tutoring System, Conference on Information Systems ECIS 2002, Gdansk, Poland
- [12] Gerhard Weber, Peter Brusilovsky, ELM-ART: An Adaptive Versatile System for Web-based Instruction, International Journal of Artificial Intelligence in Education, 2001, Vol 12, pp 351-384
- [13] <http://omweb.ida.liu.se/TeacherClient>
- [14] <http://omweb.ida.liu.se/StudentClient>

MSL Electrical Spice3 - Status and Further Development

Kristin Majetta Sandra Böhme Christoph Clauß Peter Schneider
 Fraunhofer Institute for Integrated Circuits IIS, Design Automation Division EAS
 Zeunerstr. 38, 01069 Dresden

{kristin.majetta, sandra.boehme, christoph.clauss, peter.schneider}@eas.iss.fraunhofer.de

Abstract

The Modelica Standard Library was improved by adding a package of a subset of SPICE3 models which are transformed to the Modelica language. This Spice3 library contains basic models, sources, and four semiconductor devices (diode, BJT, MOSFET level 1, resistor). Extensive tests showed the correctness of model characteristics at simple circuits. Further models already prepared will be added to the Spice3 Modelica Library later on.

Keywords: SPICE, Modelica, electronic circuit simulation, semiconductor models, netlist translator

1 Introduction

Beyond the common electric and electronic models available in the Modelica.Electrical.Analog library is has been an early aim to have Spice models for the simulation of advanced electronic circuits. Beyond the MSL two SPICE libraries already had been developed, the SPICELib [1] and the BondLib [2]. The SPICELib, which covers different complex MOSFET models, is a standalone library with its own connectors. The BondLib bases on bond graphs. It offers different levels of models related to HSPICE. At the 7th Modelica Conference the development of another SPICE library was reported as well as test issues and examples [3]. In contrast to the existing SPICE libraries the models of the new SPICE library were directly extracted from the open SPICE3 source code. These models which were improved within the ITEA research projects EURO-SYSLIB and MODELISAR, are now included into the Modelica.Electrical.Spice3 library which is available in the Modelica Standard Library version 3.2, released in October 2010.

In this paper the actual Spice3 library is presented as well as issues of testing. Furthermore, the prepared upgrading is described which concerns further models as well as a netlist translator. After all planning on a commercial extended Spice3 library is discussed.

2 The Spice3 Library of MSL 3.2

SPICE3, a simulation tool for electronic circuits, was developed in the University of California Berkeley in the nineties of the 20th century continuing the successful former version SPICE, and SPICE2 [4], [5]. It is commonly known and widely used. For a very large number of electronic circuits so called SPICE3 netlists are available, that describe the circuits for the simulation with SPICE3. The netlists are built of models of the predefined model pool of SPICE3 exclusively that contains the following models:

- Basic models (resistor, inductor, capacitor, conductor, coupled inductor, linear controlled sources)
- Semiconductor device models (different types of transistors, diode, resistor, capacitor)
- Lines (lossy and lossless transmission lines)
- Sources (different types of both voltage and current sources)

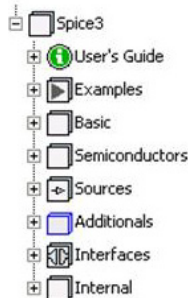
Besides SPICE3, derived simulators are known many of them are commercial, e.g. PSPICE [6], HSPICE [7]. The source code of these derivatives is, in contrast to SPICE3, not open.

The idea to have the SPICE3 models available in Modelica exists already since the beginning of the Modelica development. Between 2007 and 2010 a Spice3 library for Modelica was developed by the Fraunhofer Institute for Integrated Circuits IIS/EAS.

In the following “SPICE3” denominates Berkeley SPICE3 simulator related topics, and “Spice3” the Modelica Spice3 library.

2.1 Offered models

The models of the Modelica.Electrical.Spice3 library are thematically arranged according to the above mentioned grouping of the SPICE3 models (Picture 1). The SPICE3 line models are not yet available in Modelica. Furthermore, MSL typical packages are added (User’s Guide, Interfaces et alt.). The names of the models are composed in the way that the capital coding the model in SPICE3 (e.g. “D” for diode) is followed by an underscore and the common name of the device model, e.g. the name of the diode is *D_Diode* in the Spice3 library. This way each user independently of his knowledge of SPICE3 can indentify the model’s behavior easily.



Picture 1 Spice3 library of MSL3.2

The package *Basic* contains models of the resistor, capacitor, inductor, controlled sources and, additionally to SPICE3, the ground model. They are very similar to basic elements of the Analog library and can be used in the same way. SPICE3 netlists refer to the ground node by using the node number “0”, whereas in Modelica the ground model has to be connected.

The package *Semiconductors* contains the Mosfet level 1 (MOS1) model with the two types PMOS and NMOS. It has a set of 41 parameters and can be used for a more detailed simulation than the Mosfet transistor of the Analog package. Furthermore the BJT model, which is a bipolar transistor, was transformed into Modelica, NPN and PNP types are available. A diode model and a semiconductor resistor are also part of the package. The semiconductor resistor can either be parameterized with the usual resistance value or by its geometrical dimension (length, width) as it is in SPICE3.

The package *Sources* contains the SPICE3 voltage and current sources (constant, sinusoidal, exponential, pulse, piece wise linear and single frequency FM).

The package *Additional*s was introduced to contain important models that are not part of SPICE3 but of other SPICE derivatives. At the moment the polynomial sources of SPICE2 are part of the package. They are implemented for polynomials up to the fifth order.

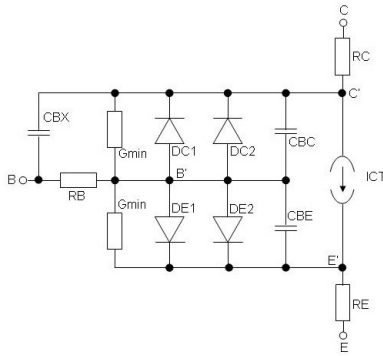
The package *Interfaces* contains only a special partial model, a twoport, that is used in the controlled sources models. The definition of the pins of the other models is used from Electrical.Analog.Interfaces. Therefore, the Spice3 models are compatible to the Electrical.Analog library and to the MSL in general.

Because the semiconductor models are very complex, many functions and data records are needed to describe the semiconductor behavior. These functions and records are collected in the *Internal* package. For the user of the Spice3 library it is not necessary to work with the *Internal* package. It is for developers only.

2.2 Principles of development

Since the SPICE3 source code is open, it is possible to extract the models directly. Years ago this was done [8]. The models were stored in a C++ library in an object oriented programming manner. This C++ library contains the exact SPICE3 models; it is tested very intensively and therefore assumed to be correct. Consequently, it was used as the base for transforming SPICE3 models into Modelica models.

In SPICE3 after linearization is calculated, in each iteration step actual representative values (R, L, G and C) of the device are written to the matrix of a linear system of equations, that basically connects the current vector with the voltage vector. Since in Modelica the terminal behavior of the model has to be described as equations this SPICE3 like linear system of equations is not necessary to be filled in by certain model functions. Instead, in Modelica a so called toplevel model is used. In Picture 2 the toplevel model of the bipolar transistor model is shown as an example, which represents the substitute circuit of the device modeled.



Picture 2 Toplevel model of bipolar transistor

The currents at the different pins (e.g. bipolar transistor: B, C, E) as well as internal currents are calculated using functions that are called in the algorithm part of the toplevel model. These functions are the main part of the model, they came from a nearly one-to-one transformation of the respectively C++ code. Also the data (parameters, variables) structure had to be rewritten into Modelica and is stored in records. The data is stored in a hierarchy in C++, because, as usual in object oriented languages, some models use the same set of parameters besides their extra own parameters. E.g. the MOS1 and MOS2 models both use nearly the same parameters. Therefore they were stored in a superior record MOS. Only the individual parameters of MOS1 and MOS2 are stored in separate records. The data structure is very complex and the number both of parameters and variables is high.

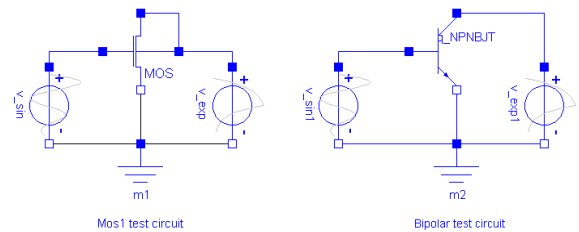
In SPICE3 two kinds of parameters are used, on the one hand the so called device parameters that are adjustable for each single device (e.g. channel length of Mos1 transistor) and on the other hand the technological parameters that are global for a group of devices (e.g. oxide thickness of Mos1 transistors). The technology parameters are set via the so called modelcard in SPICE3. That different parameter handling was transformed to Modelica by introducing modelcard records which collect the technological parameters [9]. Modelcard records are available in the Semiconductors package.

A special issue is the usage of SI units, as it is usual for the unit check in the MSL. On the one hand there are non SI units, that should not be converted to SI units in order not to use other values in the Spice3 models as it is usual in SPICE3. On the other hand, the C++-sources do not have any units at all, so they have to be added in the future.

2.3 Status of testing

Because of the high number of parameters as well as functions and especially the nonlinear behaviour, testing the Spice3 models is very complex.

To verify the simulation results of the Spice3 models SPICE3 was used as reference simulator and Dymola for the simulation of the Modelica models. Recently tests with SimulationX started. Many tests of device characteristics were developed by varying at least one parameter using a test circuit which is very simple. The device to be tested is connected to an exponential voltage source that provides the operating voltage and a sinusoidal voltage source as input. Picture 3 shows the two test circuits with the Mos1 model as well as the bipolar model.



Picture 3 Test circuits for Mos1 and Bipolar models

The SPICE3 reference results are stored in a text file and included in the test circuit simulation via the Modelica function CombiTimeTable for automatic comparison. This way it is possible to compare the results directly. Also the error between the Modelica calculation and the SPICE reference results is calculated during simulation. The comparisons are done for all currents of the pins of the models (e.g. for Mos1 the currents from the four pins Drain, Gate, Source, Bulk are compared). Since the Spice3 semiconductor models have very many parameters, a lot of tests of this kind were build (about 1800). The parameters vary from reasonable and common values to completely absurd values. The analyses of the results showed, that in most cases the simulations between Modelica and SPICE3 are in accordance for the reasonable values. In the case of absurd parameter values sometimes the results differ. However, these results are often of no practical relevance (e.g. currents of Mega ampere) but show that also in extreme cases the models work still similar. The differences possibly come from the different implementations of the simulation algorithms SPICE3 and Dymola.

These parameter tests are stored as regression tests. Their simulation instructions are stored in a separate file that runs the simulations of all the parameter tests of the according semiconductor element. The comparison results between of the currents of the Modelica Spice3 model and the imported SPICE3 currents are coded by simply “true” if correct and otherwise “false” for each pin of each test circuit, and printed into a text file for a visual check. The correctness is checked at each internal simulation step by an absolute and relative error criterion the parameters of which are individually adjusted to each test circuit. After changing something at the model, the whole regression test can be repeated for validation easily to find out possibly good or bad influence of the model changes.

Beside these relatively simple tests of model characteristics, more complex circuits were tested. However, the number of transistors in these circuits was still small. These tests were not yet done intensively, but the circuits tested showed correct results compared to SPICE3. Tests of big (transistor) circuits are still necessary. Within the test process it was recognized that numerical issues may arise at larger circuits. Also the performance of the models has to be improved in comparison to other electrical simulators, e.g. SPICE3 or Saber.

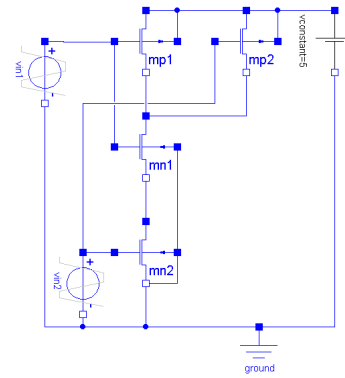
2.4 Examples

In the package *Examples* of the Spice3 library (Picture 4) eight example circuits are prepared to help the user to get an idea of the library and how to use the models.



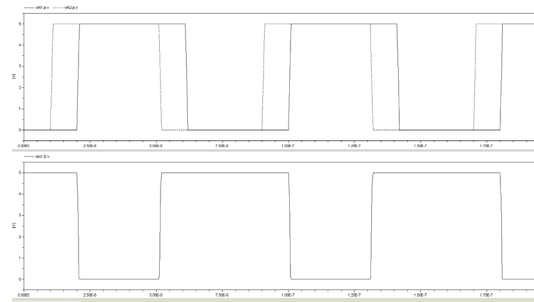
Picture 4 Package Examples of the Spice3 library

The example *Nand* is built in CMOS technology with two NMos1 and PMOS1 models (Picture 5).



Picture 5 Nand circuit of package Examples

The results are plausible (NAND function is realized) and, compared to SPICE3, correct. They are shown in Picture 6. The two upper signals are the input voltages and the lower signal is the output voltage of the NAND circuit.



Picture 6 Simulation results of the NAND circuit

To have the technology parameters available the record *modelcard* is used. In the NAND example above, the modelcards of each single transistor are filled separately. However two other possibilities are imaginable to have the modelcards available for all transistors at the same time. The first way is to make the modelcard available as instance of the circuit and give it, as a parameter, to all occurring transistors in the circuit. The second way is to define the modelcard and extend it to each transistor [3].

2.5 Open issues of the Spice3 library

The described Spice3 library was included into the MSL3.2 which was released in October 2010. The library contains a selection of the SPICE3 models. The test of the library, especially the semiconductor models, in the first instance was done via characteristic curves tests. These tests showed, that the models are working correctly. However, the test of extended electrical circuits with many transistor

models was not done yet, so information about the functionality of such real electric circuits are not yet available.

In SPICE3 it is possible to set *global parameters* to reset control options for specific simulation purpose. E.g. if GMIN is used, very small resistors are added to the circuits at certain nodes to avoid numerical difficulties. Further global parameters are:

- RELTOL (relative error tolerance)
- ABSTOL (absolute current error tolerance)
- VNTOL (absolute voltage error tolerance)
- TRTOL (transient error tolerance)
- TNOM (resetting nominal temperature)

All global parameters are described in [5]. Furthermore, SPICE3 knows *different types of analysis* like DC analysis, AC small-signal analysis, transient analysis, pole-zero analysis, small-signal distortion analysis, noise analysis and analysis at different temperatures. Since the SPICE3 models are closely related to the simulation algorithm, the models and the types of analysis are hardly clearly separable, and models are related to types of analyses. In Modelica the models and the simulation algorithm (part of the simulation tool) are separated from each other, information about the type of analysis is not part of the model but of the simulation tool. The issue is to add further types of analysis (at least AC) to Modelica and allow the models to react on the type of analysis.

In SPICE3 there are possibilities to set initial conditions to start the simulation from different states values (e.g. initial charge of a capacitance). In the actual Spice3 library for Modelica initial conditions are only available for the inductor and the capacitance model of the basic package. The *initial* parameter has to be added to the other models in the Spice3 library.

Another important point is the adding of assertions to *check parameter values* to be reasonable (e.g. temperature has to be greater than 0K, or the channel length must be positive). In SPICE3 the parameters are not checked very intensively.

A further useful feature that is not part of SPICE3 is the conditional heatport that is available in the MSL3.2. It offers the possibility of electric-thermal simulation. The heatport is a

partial model that can be included by any electrical model by inheritance. If it is included, the temperature is available within the model and the loss power has to be calculated. The heatport can be switched off. In this case the loss power flows into an internal ground and the temperature is set to the constant temperature value of the electrical model. If the conditional heatport is used, a connection between the thermal and the electrical network is available and the thermal loss power flows into the thermal network. The conditional heatport should be added in future which will change the temperature handling as modelled using fixed temperatures in SPICE3 if the heatport is switched on.

3 Further development of the MSL Spice3 package

Besides the open issues explained in section 2.5 the MSL Spice3 library will be improved by issues coming from user applications. Especially optimization of numerical issues is expected to become necessary. In this section further SPICE3 models will be described that are prepared to be added to the MSL Spice3 library as well as a netlist translator.

3.1 Planned models

To enlarge the available Spice3 model pool the Mosfet level 2 model (Mos2), the Jfet model and the coupling factor model for inductors will be added.

Mos2 model

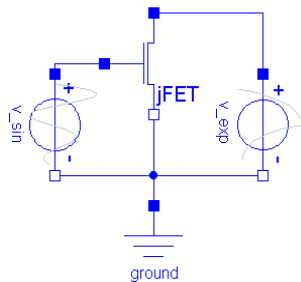
The Mos2 model is similar to the Mos1 model. Since the C++ source code is written in an object oriented manner, many of the functions and parameters of the upper Mos model were used. The Mos2 model is suitable for a smaller channel length. Actual tests of the model show, that it works correctly in principal. However a limiting precondition is, that the capacitance parameters must be set to a nonzero value to avoid numerical difficulties. The reason is, that the internal capacitance value (C_{total}) consists of two parts, the constant parameter value ($C_{parameter}$) given by the user and an internal so called Meyer capacitance value (C_{meyer}):

$$C_{total} = C_{parameter} + C_{meyer}$$

During simulation the Meyer capacitance can become zero, so the total capacitance value is also zero if the constant parameter value for the capacitance is not set. Nevertheless characteristic curve tests show that the Mos2 model is working correctly, if $C_{parameter}$ is set to a non-zero value. The capacity handling is an actual topic of research.

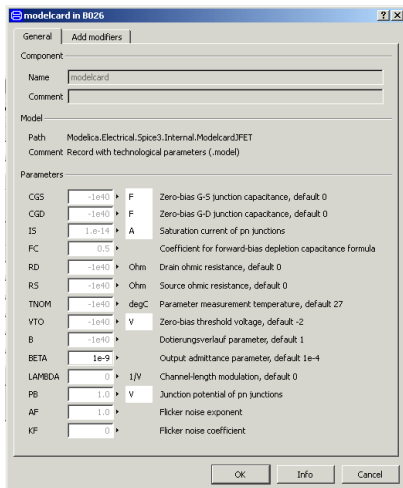
JFet model

The Junction Field Effect Transistor (Jfet, P and N Type) model is also prepared to be added to the Spice3 library. It uses the functions and parameters from the upper Fet class that is also used from the two Mosfet models of level 1 and 2. Its test is also in the first beginning comparing the characteristics using both SPICE3 and Spice3. Picture 7 shows the similar test circuit used for the characteristic curve tests of the JFet model.



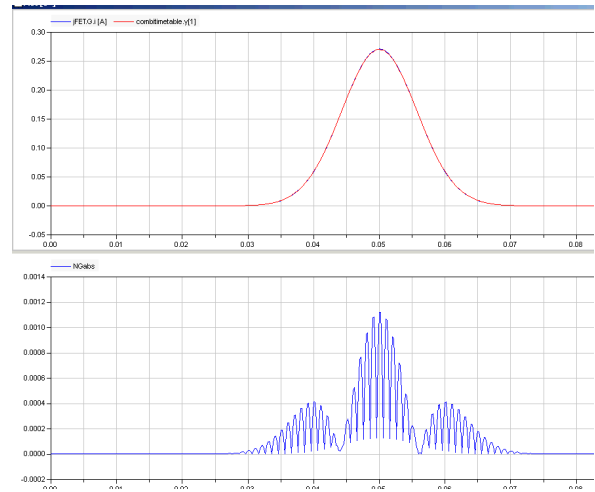
Picture 7 Characteristic curve test circuit for JFET model

As an example the influence of the transconductance parameter $BETA$ is analyzed. $BETA$ is a technology parameter that is specified in the modelcard, as it is usual in SPICE3. An impression of the JFet modelcard gives Picture 8. It can be seen that $BETA$ is given with $1e-9$ and the other parameters have their default values.



Picture 8 Jfet Modelcard

The simulation result of this circuit and the respective SPICE3 result are in accordance. As an example Picture 9 shows some simulation results.



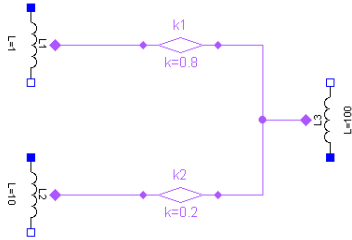
Picture 9 Simulation results of JFet test circuit

The upper picture shows the Modelica and the SPICE3 results of the gate current. These curves coincide in principle. In the lower picture the differences are shown which are in the range of mA while the maximum gate current is about 0.27A. The maximum error between the two gate currents is 0.4%. Reasons for that are the different simulation algorithms of SPICE3 and Dymola as well as the interpolation due to different step sizes.

Inductive coupling factor

In SPICE3 it is possible to model an inductive coupling between inductances via the inductive coupling factor K , which is a separate element in the SPICE3 netlist with its own element line (key letter K). It refers to two conductors which have to be defined in the netlist. To model the coupling factor in a similar way in Modelica it is suggested to introduce a new pin, the *InductiveCouplePin*, which has three variables, the inductance, the derivation of the current and, for adding the induced voltages of several couplings as a flow variable, the voltage. This pin is added to the existing inductance and to the also new introduced element *K_CoupledInductors*. Within this model, the mutual inductance M is calculated for two inductances via $M = K \sqrt{L_1 L_2}$ and with that the induced voltages $\tilde{v}_1 = M \cdot \dot{i}_2$ and $\tilde{v}_2 = M \cdot \dot{i}_1$ that are needed to complete both the voltages

$v_1 = L_1 \cdot \dot{i}_1 + M \cdot \dot{i}_2$ and $v_2 = L_2 \cdot \dot{i}_2 + M \cdot \dot{i}_1$ of the first and second inductance. This approach is still under discussion. Picture 10 gives an impression from a circuit schematic with coupled inductors.

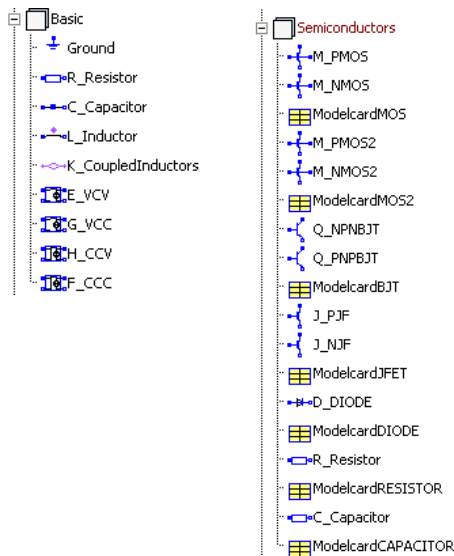


Picture 10 Circuit with coupled inductors

Semiconductor capacitance

The prepared semiconductor capacitance *C_Capacitor* is modeled like the other semiconductor devices. For technological parameters the *ModelcardCapacitor* is available which contains junction capacitances, the default width, and a narrowing factor.

Concluding the presentation of models to be added Picture 11 shows the icons of the future extended Spice3 subpackages:



Picture 11 Extended Spice3 library

3.2 Netlist translator

An important improvement will be the netlist translator [10]. It allows SPICE3 netlists to be translated to Modelica models which refer to the Models of the Modelica.Electrical.Spice3

library. The translator itself is a special Modelica model which uses essentially string function capabilities of the MSL.

As an example a SPICE3 netlist which describes a rectifier (Picture 12). The netlists file name may be *rectifier.cir*.

```
Rectifier
vsin 1 0 sin(0 5 50)

ri 1 2 0.1
di 2 3 diode area=1.5
c 3 0 30u
rl 3 0 1k

.model diode d cjo=3pF

.tran 0.001 0.1

.control
run
set options no break
.endc

.end
```

Picture 12 SPICE3 netlist of a rectifier

The translator is a Modelica model called *SpiceToModelica.mo* (Picture 13). It has to be adapted by inserting the name of the SPICE3 netlist file. Further parameters are the path to the Modelica Spice3 library, which will be used as import variable in the Modelica resulting model, the version which is prepared to enable different SPICE netlist derivatives in future, as well as the maximum number of modelcard, which is a temporary parameter of the actual solution.

```
model SpiceToModelica
"Start translation here. Enter file name,
switch to simulation mode and simulate.
The resulting .mo file will be created
in the working directory."
/-- Enter file name here: ---
parameter String fileName = "rectifier.cir"
"Name of file which shall be translated";
/-- Parameter Section ---
parameter String SpicePackage =
"Modelica.Electrical.Spice3.*"
"Location of Spice Package in Modelica Li-
brary";
parameter String version = "spice3"
"Which Spice Version shall be interpreted";
// not implemented yet
constant Integer iMax = 100
"Maximum number of modelcards in the file.";
/-- Main Program ---
protected
Boolean ok "Return value";
algorithm
ok := translate(fileName, SpicePackage,
help.upper(version), iMax);
end SpiceToModelica;
```

Picture 13 Translator call

Then the model `SpiceToModelica` has to be simulated. As a result the file `rectifier.mo` (Picture 14) contains the following Modelica model of the rectifier which can be simulated using the `Modelica.Electrical.Spice3` library:

```

model rectifier "Rectifier"
  import Modelica.Electrical.Spice3.*;

  parameter Semiconductors.ModelcardDIODE
    DIODE(CJO=3e-012);
  Sources.V_sin VSIN( VO=0, VA=5, FREQ=50);
  Basic.R_Resistor RI(R=0.1);
  Semiconductors.D_DIODE
    D1(modelcarddiode=DIODE,AREA=1.5);
  Basic.C_Capacitor C(C=3e-005);
  Basic.R_Resistor RL(R=1000);
  Basic.Ground g;

protected
  Modeica.Electrical.Analog.Interfaces.Pin
    n1;
  Modelica.Electrical.Analog.Interfaces.Pin
    n0;
  Modelica.Electrical.Analog.Interfaces.Pin
    n2;
  Modelica.Electrical.Analog.Interfaces.Pin
    n3;

equation
  connect(g.p,n0);
  connect(VSIN.p, n1);
  connect(VSIN.n, n0);
  connect(RI.p, n1);
  connect(RI.n, n2);
  connect(D1.p, n2);
  connect(D1.n, n3);
  connect(C.p, n3);
  connect(C.n, n0);
  connect(RL.p, n3);
  connect(RL.n, n0);

  annotation (uses(Modelica(version="3.2")),
    experiment(StopTime=0.1, Interval=0.001));
end rectifier;

```

Picture 14 Translated Modelica model of the rectifier circuit

Once the Modelica “netlist model” is available it can be used for simulation or adapted to further modelling. Especially connectors can be added as well as an icon to use the SPICE netlist as building block in further models. This way SPICE3 netlists can be integrated in Modelica modelling easily.

The Modelica function *translate* reads the input SPICE3 netlist as a string from the file given by the parameter `FileName`. Then preprocessing simplifies the netlist, e.g. by deleting comments. The resulting intermediate netlist is parsed to recognise several kinds of tokens which describe component names, node names, parameters and others. Since no Backus Naur Form of the SPICE3 input language exists, and the description in the manual is not as exact as expected, many tests were necessary to find out details. The SPICE3 input language for netlists is a context sensitive language therefore no prepared parser (e.g. `lex` [11]) can be applied. The used parser was constructed by taking into consideration the basic element notification of the input language as well as the specifics of many elements. The Modelica string functions were intensively used.

It is planned to add the translator which is adapted to the offered SPICE3 model pool, to `Spice3 Modelica subpackage utilities`.

4 Commercial fully extended Spice3 Library

There are still SPICE3 device models remaining which are not yet transformed to Modelica:

- MOSFET Level 3
- MOSFET Level 4 (BSIM 1)
- MOSFET Level 5 (BSIM 2)
- MOSFET Level 6 (modified Level 3)
- MESFET

These models can be transformed to Modelica in the same way as described in section 2.2. It is planned to have these models available in a fully extended Spice3 library which will be no more open source.

The transformation of further semiconductor models which are offered e.g. by HSPICE depends on the availability of the model source code. Usually the source code of models is not available in simulation tools but e.g. in publications, which need not describe identical to implementation model code. If model equations are available, and models are asked for further models can be added to the fully extended library. A challenge not discussed yet is the modeling of up-to-date MOSFET models like e.g. the EKV [12] model.

In accordance to the extended Spice3 library the netlist translator has to be adapted. If net-

lists from other SPICE simulators shall be translated which are not identical to SPICE3 netlists a thoroughly check and reprogramming of the translator will be necessary.

5 Conclusions

The MSL Electrical.Spice3 was presented which has been available since October 2010. It contains most of the basic SPICE3 models as well as low level semiconductor device models. Transistor characteristics are intensively tested at small circuits with varying parameters.

The next development step presented will add coupled inductors, further semiconductor models as well as a netlist translator which completes the library to support given SPICE3 netlists without any intermediate composition step.

It is planned to create a commercial Spice3 electronics library which offers the full SPICE3 model pool as well as models of other SPICE derivatives, furthermore additional dedicated device models.

Additional to the models a method was developed of transformation existing model descriptions, which are sequentially programmed using e.g. the C or C++ language, to Modelica. This knowledge could be applied to the transformation of further open source libraries.

Open issues in the development of electronics simulation using SPICE3 features in Modelica are

- Testing extended complex circuits
- Finding a solution of further methods of analysis (AC, sensitivity)
- Improvements to performance as well as numerical stability (e.g. homotopy operator)
- Adding the conditional heat port
- Handling of some SPICE3 options and similar issues like the initial conditions and global parameter settings

The Modelica Spice3 libraries allow more advanced electronic simulation than the Modelica.Electrical.Analog library, nevertheless electronics simulation with Modelica is to be regarded as promising, but at its initial stage.

Acknowledgement

This research was founded by the European ITEA2, research projects EUROSYSLIB and MODELISAR.

References

- [1] Urquia, A.; Martin, C.; Dormido, S.: Design of SPICELib: a Modelica Library for modeling and analysis of electric circuits. *Mathematical and Computer Modelling of Dynamical Systems*, 11(1)2005, 43-60.
- [2] Cellier, F. E.; Nebot, A.: The Modelica bond graph library. *Proc. 4th Int. Modelica Conference, Hamburg-Harburg, Germany, 1, 2005, 57-65.*
- [3] Majetta, K.; Böhme, S.; Clauß, C.; Schneider, P.: SPICE3 Modelica Library. *7th International Modelica Conference, Como, 2009.*
- [4] Nagel, L.W.: SPICE2: A computer program to simulate semiconductor circuits. *Electronic Research Laboratory Rep. No. ERL-M520, University of California, Berkeley, 1975.*
- [5] SPICE Version 3e Users Manual, 1991
- [6] Robert Heinemann: PSPICE – Einführung in die Elektronik-Simulation. Carl Hanser Verlag München, 2009. <http://www.spicelab.de/index.htm>
- [7] <http://www.synopsys.com/Tools/Verification/AMSVerification/CircuitSimulation/HSPICE/>
- [8] Leitner, T.: Entwicklung simulatorunabhängiger Modelle für Halbleiter-Bauelemente mit objektorientierten Methoden. Chemnitz, Technische Universität, Diss., 1999.
- [9] Majetta, K.: Entwicklung und prototypische Umsetzung eines Konzeptes für eine Modelica-Bibliothek von SPICE-Halbleiterbauelementen und Erarbeitung einer Teststrategie. Dresden, Berufsakademie Sachsen, Dipl., 2008.
- [10] Boneß, K.: Konzeption und prototypische Implementierung eines Übersetzers von SPICE3-Netzlisten nach Modelica basierend auf Modelica-inhärenter String-Verarbeitung. Hochschule für

Technik und Wirtschaft, Dresden, Dipl., 2011, under preparation.

- [11] Lesk, M. E.; Schmidt, E.: LEX – A Lexical Analyzer Generator. UNIX Programmer's Manual. 7th Ed., Vol. 2A. Murray Hill, New Jersey: Bell Telephone Laboratories, Inc. 1978.
- [12] Enz, C.; Krummenacher, F.; Vittoz, E.A.: An analytical MOS transistor model valid in all regions of operation and dedicated to low-voltage and low-current applications. Analog Integrated Circuits and Signal Processing 8(1995)83-114.

The New FundamentalWave Library for Modeling Rotating Electrical Three Phase Machines

Christian Kral Anton Haumer
 AIT Austrian Institute of Technology GmbH
 Mobility Department, Electric Drive Technologies
 Giefinggasse 2, 1210 Vienna, Austria

Abstract

This paper introduces the new `FundamentalWave` library which is included in the Modelica Standard Library 3.2. The presented Modelica package provides models and components of rotating electrical three phase machines. The presented electrical machine models are fully compatible with the original `Machines` library of the electrical domain but rely on the concept of the magnetic potential and magnetic flux fundamental waves. In this article, the connector concept, the components and electric machine models of the `FundamentalWave` package will be explained. Additionally, the didactic advantages and the flexibility of the proposed package in with respect to considering more enhanced and sophisticated effects will be discussed.

Keywords: Rotating electrical three phase machines, fundamental wave, time transients, Modelica Standard Library MSL

1 Introduction

Electric machines models have been introduced to the Modelica Standard Library (MSL) in 2004. Since then the package has continuously been enhanced and additional machine types and effects have been added to the `Modelica.Electrical.Machines` library (short `Machines` library). More sophisticated loss effects and a consistent thermal concept have been implemented for the library version included in the MSL 3.2. . The more sophisticated loss models include

- friction losses,
- eddy current core losses,
- stray load losses and

- brush losses,

and have been published in [1]. These loss models have been incorporated in the `Machines` library such way that they can be applied for both the original `Machines` and the new `Modelica.Magnetic.FundamentalWave` library. The main difference between the two machines packages is that `Machines` uses current, voltage and flux linkage space phasors (vectors) [2–5] whereas `FundamentalWave` applies complex vectors for physical representation of the the magnetic flux and the magnetic potential difference.

The motivation for introducing a new package for rotating electrical three phase machine was the need for modeling different effects based on physical magnetic quantities and models. An example for modeling global demagnetization effects of the permanent magnets in synchronous machines is presented in [6]. Additional applications will be discussed in section 5.

An original implementation of an electric machines library based on magnetic potential differences and magnetic flux vectors has been presented by Beuschel [7]. In this paper, however, DC and three phase electrical machine have been modeled based on a magnetic connector which is designed slightly different from the one introduced in the `FundamentalWave` library. In the original implementation of Beuschel the objective was the universal applicability of the proposed concept to different types of machines. The main focus of the `FundamentalWave` library was motivated by a clear physical interpretation of the quantities in the connector such that the library can be applied to model complex physical phenomena in electrical machines.

The `FundamentalWave` library models rely on the following assumptions:

- only symmetrical three phase induction machines are modeled

- in the current implementation only linear magnetic circuit elements are taken into account
- deep bar effects and higher harmonic spatial electro magnetic wave effects are also not considered

2 Concept and Magnetic Connector

In the package `Modelica.Magnetic.FluxTubes` the concept of magnetic ports has been introduced to model one dimensional flux tubes. The generic magnetic port of `FluxTubes`,

```
connector MagneticPort
  "Generic magnetic port"
  SI.MagneticPotentialDifference V_m
  "Magnetic potential at the port";
  flow SI.MagneticFlux Phi
  "Magnetic flux flowing into the port";
end MagneticPort;
```

consists of the magnetic potential difference as potential variable and the magnetic flux as flow variable.

In a radial field electrical machine usually only the fundamental field components is modeled. As an example, the two dimensional field distribution of a four pole asynchronous induction machine is depicted in Fig. 1. The magnetic field quantities in the air gap, e.g., the magnetic potential difference, can be approximated by a fundamental wave with respect to one pole pair, as it is presented in Fig. 2. In this figure the angle φ refers to one pole pair ($\varphi = 2\pi$) of the machine. The angle φ is also referred to as electrical angle. The magnitude and the phase shift of the fundamental wave in Fig. 2 can be represented by a phasor in the complex plane as depicted in Fig. 3. In the `FundamentalWave` library the complex magnetic potential difference and the complex magnetic flux are represented by a real and imaginary part, respectively:

```
connector MagneticPort
  "Complex magnetic port"
  Modelica.SIunits.
  ComplexMagneticPotentialDifference V_m
  "Complex magnetic potential difference";
  flow Modelica.SIunits.
  ComplexMagneticFlux Phi
  "Complex magnetic flux";
end MagneticPort;
```

The relationship between the real and imaginary parts of the phasors and the spatial waveform of the connector components is given by:

$$\begin{aligned}
 V_m(\varphi) &= \text{Re}[(V_{m,\text{re}} + jV_{m,\text{im}})e^{-j\varphi}] \\
 &= V_{m,\text{re}} \cos(\varphi) + V_{m,\text{im}} \sin(\varphi) \\
 \Phi(\varphi) &= \Phi_{\text{re}} \cos(\varphi) + \Phi_{\text{im}} \sin(\varphi)
 \end{aligned}$$

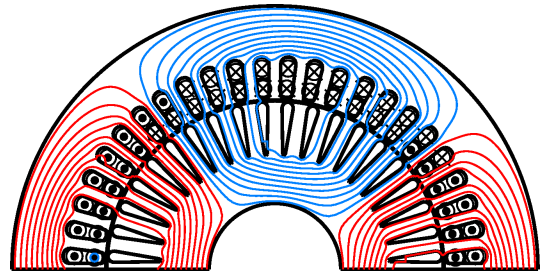


Figure 1: Magnetic field distribution of an asynchronous induction machine at no load operation

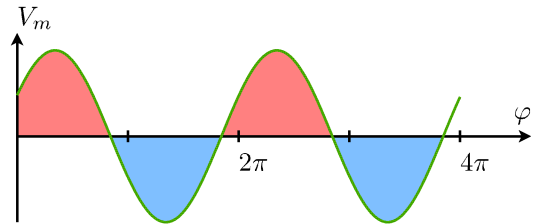


Figure 2: Fundamental wave of the magnetic potential difference

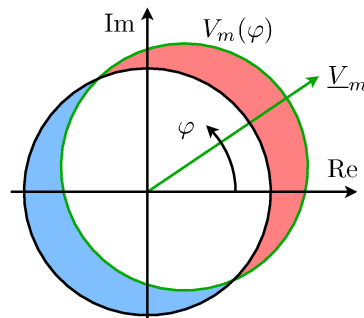


Figure 3: Complex phasor of the magnetic potential difference

It is important to note that the magnetic potential difference of the connector definition refers to the total magnetic potential difference excited by all poles of the machine.

In both the `Machines` and the new `FundamentalWave` library only spatial fundamental wave effects of spatial magnetic quantities are considered. Higher harmonic spatial wave effects due to

- the spatial reluctance distribution of the slots,
- the magneto motive force (magnetic potential difference) harmonics due to the current of the coils embedded in slots, and,
- if present, the field distribution of the permanent magnets,

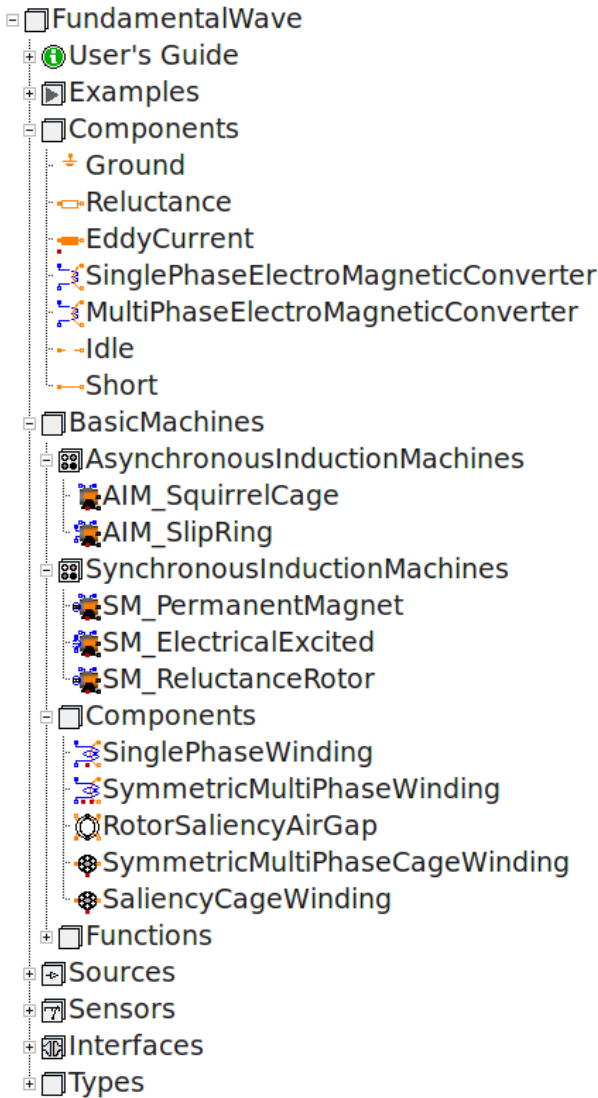


Figure 4: Structure of the `FundamentalWave` library

are not taken into account. The impact of higher harmonic time harmonics in the currents and voltages is, however, fully taken into account by the spatial fundamental wave representation.

3 Components

An overview of the packages and components included in the `FundamentalWave` library is presented in Fig. 4. In this chapter the generic components, the machine specific components and the background of the thermal super connector will be presented. Chapter 4 will then go into detail with the topology specific machine models.

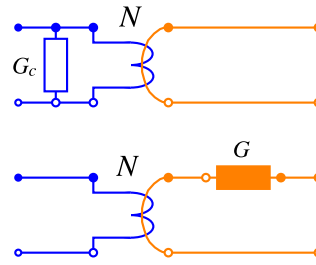


Figure 5: Equivalent consideration of eddy current losses in a multi phase electrical and a fundamental wave magnetic circuit

3.1 Generic FundamentalWave Components

The magnetic `Ground` model sets the real and imaginary part of the magnetic potential difference of the connector to zero. The complex flux components are not affected in this model.

In the `Reluctance` model a linear relationship between the magnetic potential difference and the magnetic flux components is considered. In order to take magnetic saliencies of the reluctance into account a new saliency type is defined, which consists of a `d` and `q` components. The equations of the reluctance model are:

$$\begin{aligned} (\pi/2) * V_{m, re} &= R_{m, d} * \Phi_{i, re}; \\ (\pi/2) * V_{m, im} &= R_{m, q} * \Phi_{i, im}; \end{aligned}$$

In these equations the term `pi/2` results of the averaging of the sinusoidal waveform of the magnetic flux density over one pole pair. In a fully symmetrical magnetic circuit the `d` and `q` component of the reluctance parameter `R_m` are equal. Saliency effects with unequal `d` and `q` component of the reluctance have to be applied with care, since the saliency model always refers to the specific reference frame of the connectors.

The `EddyCurrent` loss model is designed in the style of the `FluxTubes` package:

$$\begin{aligned} (\pi/2) * V_{m, re} &= G * \mathbf{der}(\Phi_{i, re}); \\ (\pi/2) * V_{m, im} &= G * \mathbf{der}(\Phi_{i, im}); \\ \text{lossPower} &= (\pi/2) * (V_{m, re} * \mathbf{der}(\Phi_{i, re}) \\ &\quad + V_{m, im} * \mathbf{der}(\Phi_{i, im})); \end{aligned}$$

In an electric machine the eddy current loss model can usually also be represented in an electrical equivalent circuit as depicted in Fig. 5. The equivalent representation of the two circuits, however, relies on a symmetrical electro magnetic coupling as it is explained in the following paragraph. For an `m` phase electrical circuit the parameter relationship

$$G = (\pi/2) * G_c / N^2$$

applies.

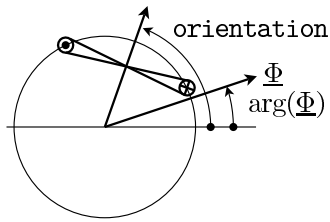


Figure 6: Orientation of a single phase coil and complex magnetic flux phasor

The `SinglePhaseElectroMagneticConverter` considers Ampere's law and the induction law for a single phase coil. Model parameters are the effective number of turns and the orientation, both with respect to the fundamental wave:

```
parameter Real effectiveTurns
  "Effective number of turns";
parameter Modelica.SIunits.Angle orientation
  "Orientation of the resulting
  fundamental wave V_m phasor";
final parameter Complex N =
  effectiveTurns * Modelica.ComplexMath.exp(
    Complex(0,orientation))
  "Complex number of turns";
```

Such a single phase coil is depicted in Fig. 6 and the relevant model equations are:

```
V_m = (2.0/pi) * N * i;
-v = Modelica.ComplexMath.real(
  Modelica.ComplexMath.conj(N)
  *Complex(der(Phi.re), der(Phi.im)));
```

In the `MultiPhaseElectroMagneticConverter`, incorporating m phases, an array of m single phase electro magnetic converters is instantiated and connected with the multi phase electrical connectors. The magnetic ports of the single phase electro magnetic converters are series connected, since the same flux applies to each coupling element.

3.2 Machine Specific Models

For modeling electric machines some more specific models are provided. These models are single and symmetrical multi phase windings including

- winding resistance including thermal connector,
- leakage inductance and field, respectively,
- zero inductance,
- core losses associated with the respective winding including thermal connector, and the
- electro magnetic converter.

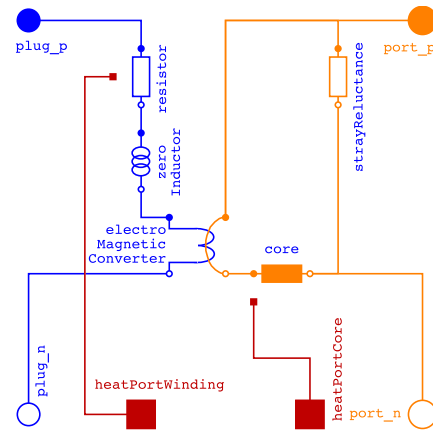


Figure 7: Symmetrical multi phase winding

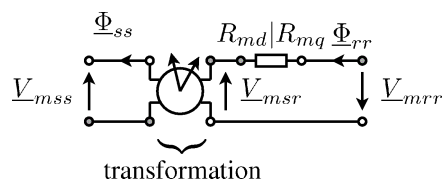


Figure 8: Stator and rotor fixed complex fluxes and magnetic potential differences of the air gap model

Additional machine specific models are the air gap model (with rotor saliency) as well as the symmetrical and salient rotor cage windings.

The `SymmetricalMultiPhaseWinding` model is depicted in Fig. 7. The stray field and the core losses of the winding are considered in the magnetic domain. Since the modeled stray field implies an ideal coupling of the m electrical phases, the zero inductance of the machine has to be considered separately [8]. The thermal connectors of the winding resistor and core loss model are externally available.

In the `RotorSaliencyAirGap` model different physical effects are taken into account. First, the stator and rotor magnetic ports have different fundamental wave rotational frequencies as they refer to different reference frames, see Fig. 8:

```
// Stator flux, stator fixed
port_sp.Phi = Phi_ss;
// Rotor flux, rotor fixed
port_rp.Phi = Phi_rr;
// Stator magnetic potential difference,
// stator fixed
port_sp.V_m - port_sn.V_m = V_mss;
// Rotor magnetic potential difference,
// rotor fixed
port_rp.V_m - port_rn.V_m = V_mrr;
```

The quantities that either refer to the stator or rotor reference frame are transformed by means of a rotor that is derived from the electrical angular difference

gamma between mechanical stator and rotor flange— thus multiplied with the number of pole pairs p :

```

gamma = p*(flange_a.phi-support.phi);
rotator = Modelica.ComplexMath.exp(
    Complex(0,gamma));
// Stator flux, rotor fixed
Phi_sr = Phi_ss
    * Modelica.ComplexMath.conj(rotator);
// Stator magnetic potential difference,
// rotor fixed
V_msr = V_mss
    * Modelica.ComplexMath.conj(rotator);
    
```

Second, the magnetic reluctance of the air gap due to rotor saliency is taken into account, since the relationships between the components of the fluxes and the magnetic potential differences are considered with respect to the rotor reference frame.

```

// Total magnetic potential difference
// is the sum of the stator and rotor
// magnetic potential difference
(pi/2.0) * (V_mrr.re + V_msr.re)
    = Phi_rr.re*R_m.d;
(pi/2.0) * (V_mrr.im + V_msr.im)
    = Phi_rr.im*R_m.q;
    
```

Cross coupling effects are not considered in this model.

All the electric machine models provided by the `FundamentalWave` library do not take the reluctances of the stator and rotor teeth and yoke into account. Therefore, the total main field reluctance of the machine is seen as an equivalent air gap reluctance. Third, the inner torque of the air gap is calculated:

```

tauElectrical =
    - (pi*p/2.0)*(Phi_ss.im * V_mss.re
        - Phi_ss.re * V_mss.im);
flange_a.tau = -tauElectrical;
support.tau = tauElectrical;
    
```

The `SymmetricMultiPhaseCageWinding` and `SaliencyCageWinding` are designed in a similar way as the symmetric winding models. The main difference is the inherent short circuit of the cage winding and thus the cage models have only magnetic and thermal connectors, respectively. An example of a symmetric rotor cage model is depicted in Fig. 9. In this implementation the rotor cage is modeled as an equivalent three phase winding—which is already implied by the machine parameters rotor resistance R_R and rotor leakage inductance $L_{r\sigma}$, which both refer to an equivalent stator winding. Rotor core losses are currently not considered in the cage winding models.

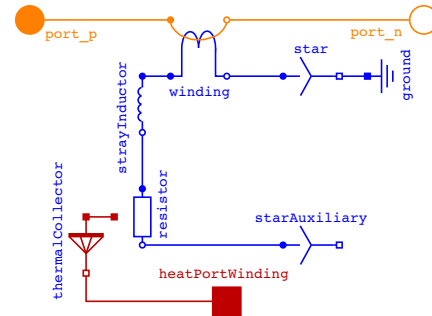


Figure 9: Symmetric rotor cage model

3.3 Thermal Super Connectors

For all the electric machine models thermal super connector are introduced. The super connectors are only implemented in the `Machines` library, since both the `Machines` and the `FundamentalWave` library use the same connector definitions. The super connector contains instances of thermal connectors. Each thermal connector is associated with the actual temperature and heat flow of one particular loss effect. Since the different machine models have different topologies, different super connector definitions are used. The common heat ports of all induction machine models are summarized in the partial model connector:

```

partial connector
PartialThermalPortInductionMachines
    "Partial thermal port
    of induction machines"
parameter Integer m=3 "Number of phases";
Modelica.Thermal.HeatTransfer.Interfaces.
    HeatPort_a heatPortStatorWinding[m]
    "Heat port of stator windings";
Modelica.Thermal.HeatTransfer.Interfaces.
    HeatPort_a heatPortStatorCore "Heat port
    of (optional) stator core losses";
Modelica.Thermal.HeatTransfer.Interfaces.
    HeatPort_a heatPortRotorCore "Heat port
    of (optional) rotor core losses";
Modelica.Thermal.HeatTransfer.Interfaces.
    HeatPort_a heatPortStrayLoad "Heat port
    of (optional) stray losses";
Modelica.Thermal.HeatTransfer.Interfaces.
    HeatPort_a heatPortFriction "Heat port
    of (optional) friction losses";
end PartialThermalPortInductionMachines;
    
```

The basic thermal super heat port contains a connector for the stator winding (copper losses), the stator core (currently only stator eddy current losses), the rotor core (currently not utilized), stray load losses and the friction losses. Each connector used for the particular induction machine models extends from this partial connector definition and adds machine specific parameters and heat ports. For example, the thermal super

connector definition for permanent magnet induction machines is:

```

connector ThermalPortSMPM
  "Thermal port of synchronous induction
  machine with permanent magnets"
  extends Machines.Interfaces.
    InductionMachines.
    PartialThermalPortInductionMachines;
  parameter Boolean useDamperCage
    "Enable / disable damper cage";
equation
  Modelica.Thermal.HeatTransfer.Interfaces.
  HeatPort_a heatPortRotorWinding
  if useDamperCage
    "Heat port of damper cage (optional)";
  Modelica.Thermal.HeatTransfer.Interfaces.
  HeatPort_a heatPortPermanentMagnet
  "Heat port of permanent magnets";
end ThermalPortSMPM;
    
```

Since the permanent magnet synchronous machines are implemented with an optional damper cage, the boolean parameter `useDamperCage` has to be utilized in the associated heat port definition.

4 Electric Machine Models

In the `FundamentalWave` library five different induction machine models are provided. Each of these models extends from the partial machine model which is depicted in Fig. 10. Each machine model consists of electrical, mechanical and thermal connectors. The stator windings are accessible by a positive and negative multi phase plug with $m=3$ phases, representing the begin and the end of the three stator windings, respectively. Each model that dissipates losses is equipped with a thermal heat port. Each of these thermal heat ports is connected to an internal thermal port (super connector). In order to consider heat flow and temperature exchange with external thermal models an optional external thermal port is provided. Dependent on a boolean parameter `useThermalPort` either the external thermal port is used or the losses are dissipated to an internal constant temperature ambient. The shaft end of the machine is represented by the rotational connector `flange`. By default, the stator is fixed and thus the rotational connector `support`, representing the stator housing, is not accessible. Alternatively, if the parameter `useSupport` is set to `true`, the rotational connector of the stator housing can be connected with an external mounting model.

The electrical plugs are connected with a model of the stray load losses and a symmetrical model of the stator winding (Fig. 7). The stray load loss model senses the

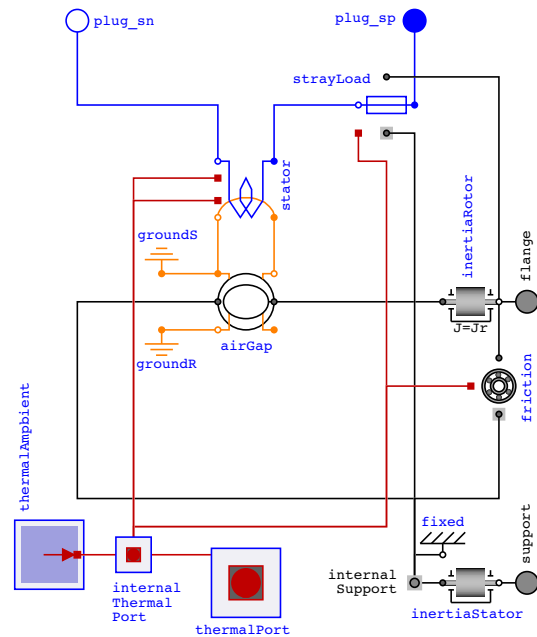


Figure 10: Partial basic induction machine model

actual current and generates mechanical losses proportional to the current and angular velocity. The magnetic ports of the stator winding model are directly connected to stator ports of the air gap model. The inertias of the stator and rotor, respectively, are connected to all the stator and rotor specific rotational connectors of the air gap, stray load loss and friction model, respectively.

The machine parameters of the `Machines` and the `FundamentalWave` library are identical except for one parameter, the effective number of stator turns, `effectiveStatorTurns`, which is solely needed in the `FundamentalWave` library. This parameter does not affect the operational behavior of the machine but scales the magnetic potential difference and magnetic flux. The fundamental wave connector quantities can therefore only represent the actual physical quantities of the machine, if the real effective number of turns is provided.

4.1 Asynchronous Induction Machines

A squirrel cage and slip ring induction machine model are provided in the `FundamentalWave` library. In the squirrel cage model the magnetic rotor ports of the air gap model are connected with a symmetrical cage model as depicted in Fig. 11. The slip ring induction machine models has a regular three phase winding in the rotor, see Fig. 12. In the current implementation, the rotor core losses are not taken into account.

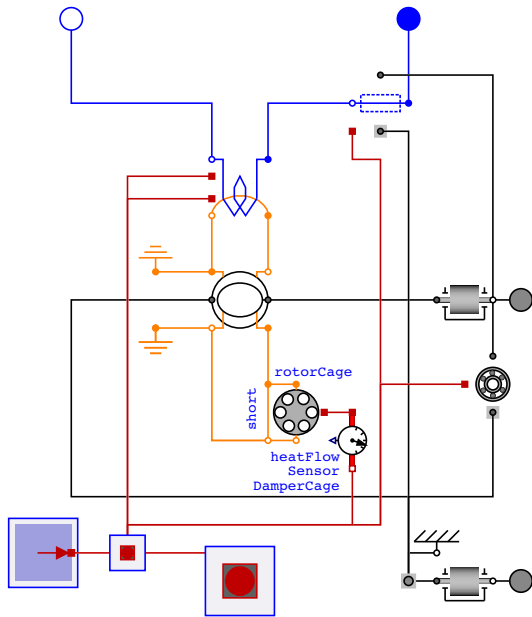


Figure 15: Synchronous machine with reluctance rotor

4.2 Synchronous Induction Machines

Three synchronous machine models with electrical excitation, permanent magnet excitation and without excitation but a reluctance rotor are available in the `FundamentalWave` library. Each of these machine models is equipped with an optional damper cage. The damper cage parameters include different rotor resistance and rotor leakage inductance parameters in the direct (d) and quadrature (q) axis. If the damper cage is disabled, a magnetic short is connected instead of the damper cage, as show in Fig. 13–15. In order to handle the cage temperature of the super connector in case of the disabled damper cage, a heat flow sensor is uses which can set its connector temperature for consistency reasons. The permanent magnet synchronous machine model depicted in Fig. 13 has a magnetic potential difference source in the rotor representing the permanent magnet. The reluctance of the magnet is inherently represent by the salient main field inductances of the direct (d) and quadrature (q) axis, L_{md} and L_{mq} , respectively. The synchronous machine with electrical excitation has a single phase electrical winding (Fig. 14) and a brush model representing the slip rings of the excitation circuit. In the reluctance synchronous machine model of Fig. 15 no additional excitation is present. The torque generated in this synchronous machine is solely due to rotor saliencies, i.e., $L_{md} \neq L_{mq}$.

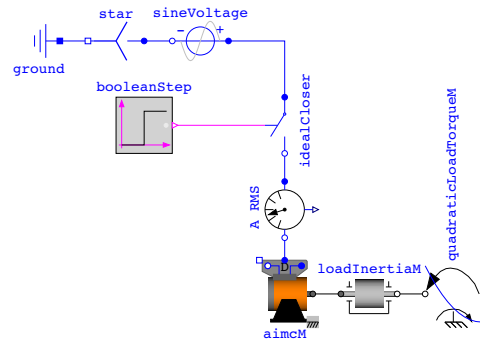


Figure 16: Asynchronous induction machine started direct on line

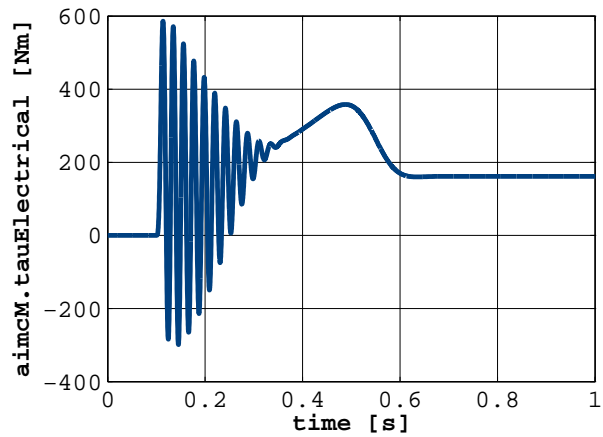


Figure 17: Torque of an asynchronous machine with squirrel cage rotor started directly on line

4.3 Example

An example of an asynchronous induction machine started direct on line is presented in Fig. 16, comparable to the included one in the `Examples` package of the `FundamentalWave` library. A delta connected induction machine is connected to a stiff voltage supply after closing a switch at 0.1 s. Even if this is not presented here in this paper: the torque developed by the induction machine during starting (Fig. 17 and 18) is exactly matching the results that are obtained by substituting the asynchronous induction machine by a model from the `Machines` library. So the models are fully compatible with respect to their operational behavior. For the investigated machine the effective number of stator turns was set to 25 which results in the rotor flux results presented in Fig. 19. Immediately after closing the switch at 0.1 s the rotor frequency is equal to the supply frequency (50 Hz). After fully accelerating the machine, the slip frequency of the rotor flux is approximately 2 Hz.

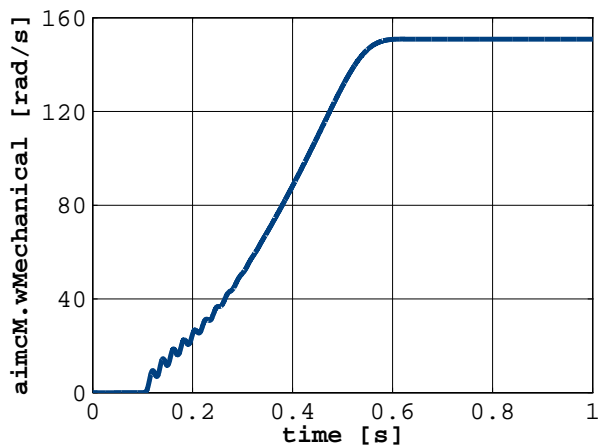


Figure 18: Angular mechanical velocity of an asynchronous machine with squirrel cage rotor started directly on line

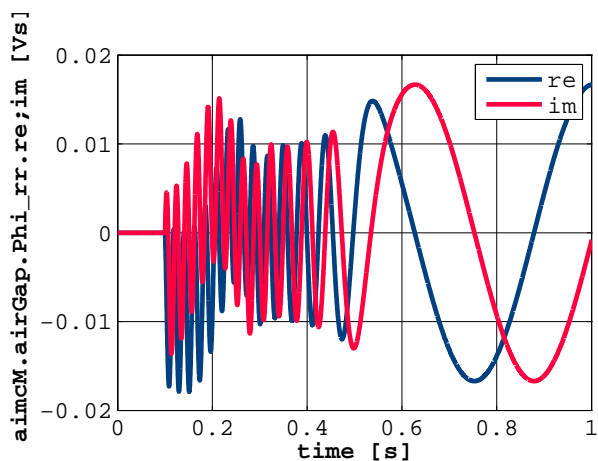


Figure 19: Real and imaginary part of the rotor flux of an asynchronous machine with squirrel cage rotor and 25 stator turns started directly on line

5 Didactic Aspects and Expandability

From a didactic point of view the `FundamentalWave` electric machines library allows a valuable insight in the concepts of electro mechanical power conversion. Since fundamental wave magnetic domain models are added to the MSL, the magnetic potential and flow variables of an electric machine do have a clear and physical representation. For each component and physical effect in an electrical machine there is a particular model representing this effect. For example, in an induction machine with slip ring rotor the machine basically consists of a stator winding, an air gap and a rotor winding. The total reluctance of the magnetic circuit of the machine is represented by the main field

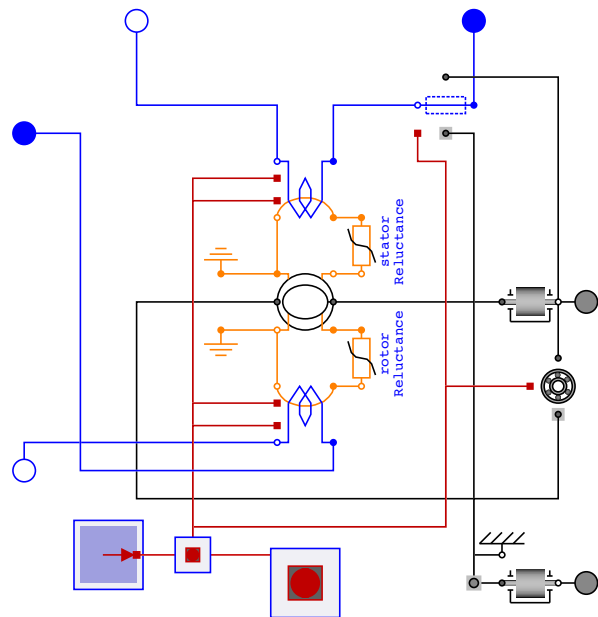


Figure 20: Possible model of an asynchronous induction machine with nonlinear stator and rotor core reluctances

inductance of the air gap model. The reluctances of the stator and rotor teeth and yoke are currently not explicitly modeled.

With the `FundamentalWave` library the model can certainly be adapted such way that the nonlinear magnetic properties of the stator and rotor core can be considered, as it is proposed in Fig. 20. In such a model the air gap reluctance has to be adapted such way that it only represents the total main field reluctance minus the stator reluctance minus the rotor core reluctance. In such a model the different section of the magnetic main field—stator core, air gap, rotor core—would be represented by the respective magnetic reluctance models.

The temperature dependent ohmic losses of the stator and rotor winding—if present—as well as the stator core losses are inherently taken into account in the winding models. Friction and stray load losses are considered by separate losses. If either of these losses should be modeled in more sophisticated way, e.g., extending the core losses by hysteresis losses, then only the respective loss model has to be replaced and adapted due to the clearly object oriented structure of the components applied in the machine models.

Additional physical effects such as the deep bar effect can be easily modeled by replacing the respective cage model by a more advanced one. In the same way, the entire topology of an asymmetric squirrel cage could be modeled by replacing the three phase symmetrical

topology by a more sophisticated one [9].

6 Conclusions

This paper gives an overview of the `FundamentalWave` library which is included in the Modelica Standard Library 3.2. The basic idea of the complex magnetic potential difference and the complex magnetic flux is introduced and the basic components and equations are presented. After discussing the machine specific components the common partial machine model and the different asynchronous and synchronous induction machines are presented. From a modeling point of view the basic structure of the fundamental wave electric machine models is very clear and plausible. Each model represents a distinct physically effect and can thus easily be replaced by a more sophisticated model in order to consider additional physical effects.

References

- [1] A. Haumer, C. Kral, H. Kapeller, T. Bäuml, and J. V. Gragger, “The AdvancedMachines library: Loss models for electric machines”, *Proceedings of the 7th Modelica Conference*, pp. 847–854, 2009.
- [2] G. J. Retter, *Matrix and Space-Phasor Theory of Electrical Machines*, Akademiai Kiado, Budapest, 1987.
- [3] C. Kral, “Derivation of the space phasor equations and the required parameters of a squirrel cage induction machine with faulty rotor bars”, *Conference Proceedings of the International Symposium on Diagnostics for Electrical Machines, Power Electronics and Drives, SDEMPED*, pp. 395–400, 1999.
- [4] J. Stepina, “Raumzeiger in Matrizendarstellung in der Theorie der elektr. Maschine”, *Archiv fuer Elektrotechnik*, vol. 58, pp. 91–97, 1976.
- [5] H. Kleinrath, *Stromrichtergespeiste Drehfeldmaschinen*, Springer Verlag, Wien, 1980.
- [6] C. Kral, R. Sprangers, J. Waarma, A. Haumer, O. Winter, and E. Lomonova, “Modeling demagnetization effects in permanent magnet synchronous machines”, *Proceeding of the XIX International Conference on Electrical Machines 2010*, 2010.
- [7] M. Beuschel, “A uniform approach for modelling electrical machines”, *Modelica workshop*, pp. 101–108, 2000.
- [8] H. Späth, *Elektrische Maschinen, Eine Einführung in die Theorie des Betriebsverhaltens*, Springer Verlag, Berlin – Heidelberg – New York, 1973.
- [9] C. Kral, A. Haumer, and F. Pirker, “A modelica library for the simulation of electrical asymmetries in multiphase machines - the extended machines library”, *IEEE International Symposium on Diagnostics for Electric Machines, Power Electronics and Drives, The 6th, SDEMPED 2007, Cracow, Poland*, pp. 255–260, 2007.

A Driver Model for Virtual Drivetrain Endurance Testing

Tilman Bunte
 German Aerospace Center (DLR)
 Institute of Robotics and Mechatronics, Germany
Tilman.Buente@dlr.de

Emmanuel Chrisofakis
 Daimler AG
 Stuttgart, Germany
emmanuel.chrisofakis@daimler.com

Abstract

Starting from an assumed vehicle path on a given road section we derive the formulae for the calculation of an appropriate reference speed profile tabled over road arc length. Together with a speed error feedback we thus emulate what a real driver does while driving and scheduling his actions on throttle and brake pedal. The resulting driver model may be used for automatic speed control in vehicle dynamics simulation. The application addressed here is software in the loop simulation for virtual drivetrain endurance testing at Daimler AG. A prototypical Modelica implementation was made at DLR and tested with a simple longitudinal vehicle dynamics model. Finally, we discuss the experiences with the reference speed profiles made in the industrial practice.

Keywords: driver model; automatic gearbox testing; reference vehicle speed profile; software in the loop (SiL); virtual drivetrain endurance testing.

1 Introduction

For the virtual endurance test of automatic gearboxes realistic and repeatable load collectives are searched for. Therefore, a given road is assumed in terms of slope, crossfall, curvature, road adhesion coefficient, and speed limits along the path of the road centerline. Adequate input signals for throttle and brake pedal are needed to drive a total vehicle model along the road while imitating realistic driver behavior. The driver model task means providing suitable pedal position signals. In our approach, the driver model is split into two sequential subtasks. Firstly, a vehicle speed profile along the road arc length is calculated regarding the road conditions with a sufficient preview. This reference speed profile is supposed to approximate a speed profile which a driver (usually unconsciously) forms in his mind yielding a set point for subsequent speed control. Speed profiles have already been used in the context of various driver speed assistance systems such as [1], [2]. The

idea is continued and re-engineered here to obtain an adequate complexity of the solution for the given problem.

According to the split subtask notion, the second subtask is accomplished by using feedback of the speed error to desired longitudinal vehicle acceleration. Based on the latter appropriate gas and brake pedal positions can be determined. Meaningful parameterization allows for assigning typical driver types like *cautious*, *normal*, *sportive* or *risky*. The driver model is implemented in Modelica; the speed profile calculation is done at initialization time.

The paper is organized as follows. The assumptions and the theoretic background of the speed profile generation are described in section 2. Section 3 is dedicated to the implementation of a speed feedback controller. Some simulation results with the model depicted in Figure 1 are shown in section 4. Experimental results from virtual automatic gearbox test runs are presented in section 5 including a report on one's experiences.

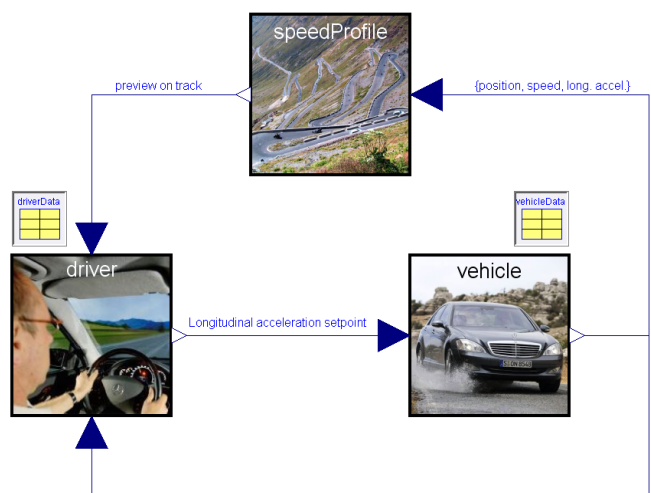


Figure 1: Total Modelica model for driver model evaluation

2 Speed profile generation

The derivation of a reference speed profile is according to the following conception: At the end of a preview horizon the vehicle should come to a standstill. The preview horizon may be set arbitrarily a certain distance ahead or e.g. formed by the end of the current visual range, the next road junction, and/or by an obstacle. On the way to this stop the vehicle's speed is scheduled to be maximal, however, such that all traffic regulations and physical limitations are met with certain margins. These include the limited lateral acceleration in curves, reduced deceleration capability while downhill or curve riding due to combined longitudinal/lateral tire forces, speed limits, and so on. The margins are adapted according to the driver type. In summary: Like real drivers do, the speed is scheduled virtually along the reverse path starting from a limitation arising ahead.

2.1 Assumptions

The road definition is assumed to be given in terms of slope $\partial z(s)/\partial s$, crossfall $\partial z(s)/\partial w$, curvature $\rho(s)$, road adhesion coefficient $\mu(s)$, and speed limits as functions of a single parameter being the road arc length s . The variable's dependencies of s (also denoted *position* in the sequel) are omitted in the formulae as from now.

The road position $[x, y, z]^T$ and the heading angle φ belonging to any value of the arc length s and any lateral displacement w from the road centerline can be calculated by numeric integration based on the following ordinary differential equations

$$\frac{\partial \varphi}{\partial s} = \rho, \quad \frac{\partial x}{\partial s} = \cos(\varphi), \quad \frac{\partial y}{\partial s} = \sin(\varphi) \quad (1)$$

and adequate start conditions. Road slope and crossfall should be so small such that errors from linearization of associated trigonometric functions are negligible. A basic supposition adopted here is that the total horizontal force $|F_{sw}|$ transmitted between the collectivity of all tires and the road is limited isotropically,

$$|F_{sw}| = \sqrt{F_s^2 + F_w^2} \leq \mu \cdot m \cdot g, \quad (2)$$

where g is the gravitational acceleration, F_s is the force in travel direction, and F_w represents the lateral force.

The disposition of the driver to utilize the physical force limits in longitudinal or lateral direction is reflected by the driver behavioral parameters κ_s and κ_w , respectively, each with $0 \leq \kappa_{s,w} \leq 1$ and specific

values depending on the driver type. Evolving from (2), the expression

$$\frac{\sqrt{\left(\frac{F_s}{\kappa_s}\right)^2 + \left(\frac{F_w}{\kappa_w}\right)^2}}{\mu \cdot m \cdot g} \leq 1 \quad (3)$$

is denoted the *driver related degree of utilization of force transmission quota* which usually is persistently changing while driving. The associated inequality is the fundamental relation [1] later used for the calculation of speed profiles.

2.2 Forces acting on the vehicle

For the calculation of the reference speed profile the vehicle is considered a point mass. Therefore, vehicle dynamics such as yaw, roll, pitch and heave motion plus their effect on the tire forces are neglected. The speed of the point mass vehicle is $v = ds/dt$, the longitudinal force may be expressed as

$$F_s = m \left(\dot{v} + \lambda \cdot v \cdot |v| + g \cdot \left(k_R(v) - \frac{\partial z}{\partial s} \right) \right) \quad (4)$$

where $k_R(v)$ is the vehicle's rolling resistance coefficient and

$$\lambda = \frac{\rho_L c_w A}{2 \cdot m} \quad (5)$$

is a parameter related to aerodynamic drag defined only for abbreviation of math terms. Here, ρ_L is the air density, c_w is the drag coefficient, A is the face surface, and m is the total mass of the vehicle.

The lateral force is

$$F_w = m \left(\rho \cdot v^2 + \frac{\partial z}{\partial w} \cdot g \right). \quad (6)$$

Note that the point mass assumption does not hold for highly dynamic manoeuvres which may result from *risky* driver behaviour. In this case, it can not be guaranteed that the real vehicle would still be able to follow the speed profile.

2.3 Constraints on longitudinal dynamics

The calculation of reference speed profiles is determined by a set of constraints on the longitudinal dynamics of the vehicle which are presented below.

Static speed limits

An upper static bound on the speed is obtained when solving (3) for v after inserting (6) and $F_s = 0$:

$$v \leq \sqrt{\frac{\kappa_w \cdot \mu \cdot g}{|\rho|} - \frac{g}{\rho} \cdot \frac{\partial z}{\partial w}} \quad (7)$$

This is the local maximum speed without making skidding off the road in a curve. Or, to be more precise, the portion of it the driver is accepting.

Another bound reflects the collectivity of all conceivable speed limitations such as legal speed limits, deliberate speed reduction or any other arbitrary speed constraint:

$$v \leq \frac{\kappa_f}{\kappa_v} \cdot v_{speedlimit} \quad (8)$$

The behavioral parameter κ_f reflects the driver's disposition towards this constraint category. The value $\kappa_f = 1.1$ means that the driver is ready to exceed speed limits by ten percent. The parameter κ_v in (8) will be cancelled later (see (23)) and is of no relevance here. The static upper speed limit, in summary, is the smaller of the two limits calculated by (7) and (8):

$$v \leq v_{max,stat}$$

with

$$v_{max,stat} = \min \left\{ \sqrt{\frac{\kappa_w \cdot \mu \cdot g}{|\rho|} - \frac{g}{\rho} \cdot \frac{\partial z}{\partial w}}, \frac{\kappa_f}{\kappa_v} \cdot v_{speedlimit} \right\} \quad (9)$$

Acceleration limits

The limited engine power P_{max} imposes an upper bound on the acceleration. Depending on the driver type the power limit is exploited by a fraction κ_p with $0 \leq \kappa_p \leq 1$, thus

$$F_s \cdot v \leq \kappa_p \cdot P_{max} \quad (10)$$

holds. After insertion of (4) and solving for the acceleration we get

$$\dot{v} \leq \dot{v}_{max,P} = \frac{\kappa_p \cdot P_{max}}{v \cdot m} + c \quad (11)$$

with

$$c = -\lambda \cdot v \cdot |v| - g \cdot (k_R(v) - \partial z / \partial s). \quad (12)$$

A valid interval for the vehicle's acceleration can be obtained from transformation of (3) and consideration of (4), (6), (11), and (12):

$$c - d \leq \dot{v} \leq c + e \quad (13)$$

with

$$d = g \frac{\kappa_s}{\kappa_w} \sqrt{\kappa_w^2 \cdot \mu^2 - \left(\frac{\rho \cdot v^2}{g} + \frac{\partial z}{\partial w} \right)^2} \quad (14)$$

and

$$e = \min \left\{ d, \frac{\kappa_p \cdot P_{max}}{v \cdot m} \right\}. \quad (15)$$

2.4 Numeric speed profile calculation formulae

On a sufficiently small section $s_0 \leq s \leq s_1$ of the road path where the longitudinal acceleration can be assumed constant the following equation holds:

$$s_1 = s_0 + v_0 \cdot \Delta t + \frac{\dot{v}}{2} \Delta t^2 \quad (16)$$

Here, Δt is the time needed to drive along the road section and v_0 is the initial speed at $s = s_0$. Moreover, the speed v_1 when reaching $s = s_1$ is

$$v_1 = v_0 + \dot{v} \cdot \Delta t. \quad (17)$$

Depending on whether v_0 or v_1 is given, after elimination of Δt from the set (16), (17) we obtain meaningful solutions for the other variable

$$v_1 = \sqrt{v_0^2 + 2 \cdot \dot{v} \cdot (s_1 - s_0)} \quad (18)$$

or

$$v_0 = \sqrt{v_1^2 - 2 \cdot \dot{v} \cdot (s_1 - s_0)}. \quad (19)$$

This allows for a simple numeric integration algorithm (explicit Euler) for the calculation of reference speed profiles along the arc length s

$$v_{(k+1)} = \sqrt{v_{(k)}^2 + 2 \cdot \dot{v}_{(k)} \cdot (s_{(k+1)} - s_{(k)})} \quad (20)$$

or

$$v_{(k-1)} = \sqrt{v_{(k)}^2 - 2 \cdot \dot{v}_{(k)} \cdot (s_{(k)} - s_{(k-1)})} \quad (21)$$

for the reverse direction, respectively. For this purpose it is required that the road path information is given with sufficiently high resolution along s such that the assumption of constant longitudinal acceleration between the grid points is justified.

2.5 Numeric maximal speed profile calculation

The *maximal speed profile* denotes the speed profile $v_{max}(s)$ along a considered road section exhibiting the maximum possible speed at all positions s while respecting the following constraints:

- The start speed at the road section beginning is $v_{max}(s_{start}) = v_{start}$.
- The final speed at the road section end is $v_{max}(s_{end}) = v_{end}$.
- At every position $s_{start} \leq s \leq s_{end}$ the inequalities (3), (9) with $\kappa_v = 1$, and (13) hold.

Hence, the maximal speed profile is a candidate for a reference speed along the considered road section to be used for speed control. It may, of course, be further processed according to one's needs.

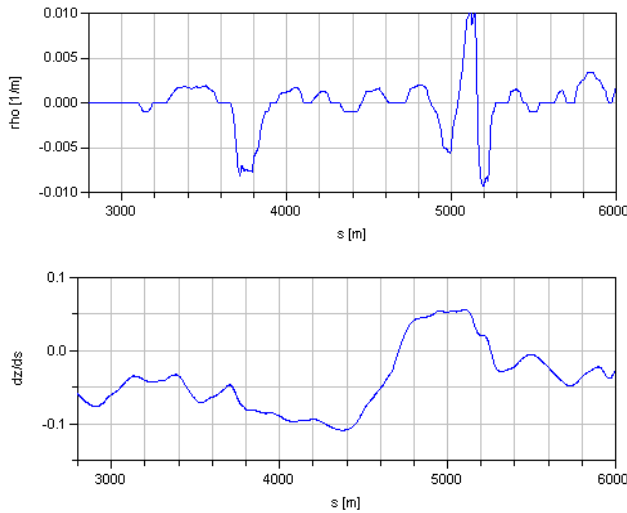


Figure 2: The road definition used for illustration in sections 2-4: $\rho(s)$ and $\partial z(s)/\partial s$ as displayed; $\mu = 1$, $\partial z(s)/\partial w = 0$, $v_{speedlimit} = 41.7\text{m/s}$.

In the sequel the procedure for calculating a tabled representation of the maximal speed profile is presented. In multiple steps the speed profile is reduced by considering new constraints at each time. See Figure 3 for illustration. The underlying road definition is given with Figure 2. The *normal driver* type was chosen; see Table 1 in section 4.2.

1. The static upper speed limit $v_{max,stat}$ is calculated using (9) for all given sampling points of s .
2. Starting from the end of the road section s_{end} an interim profile $v_{max,back}$ is calculated. Therefore, the recursive formula (21) is applied over all sampling points of s and $v_{max,back}(s_{end}) = v_{end}$ is used as start value. With each integration step the value of \dot{v} is

set to $\dot{v} = c - d$ being the maximum deceleration (i.e. minimum acceleration) according to (13). Therefore, in (12) and (14) the current values for all varying quantities are inserted. During the recursive procedure $v_{max,back}$ must be always limited to the static upper speed limit $v_{max,stat}$. The resulting interim speed profile provides a necessary condition such that the vehicle starting with $v_{max,stat}$ at any position can decelerate down to $v(s=s_{end}) \leq v_{end}$ while always respecting the inequalities (3), (9).

3. Not only when braking, also when accelerating the constraints must be fulfilled. Therefore, the previous step is repeated, however, in forward direction resulting in a new interim speed profile $v_{max,forw}$. Formula (20) is used for recursive integration from the start value v_{start} at s_{start} . The acceleration is set to its current maximum value $\dot{v} = c + e$. Note, that $v_{max,forw}$ mustn't exceed the previously calculated $v_{max,back}$ in order to keep that information.
4. The finite difference equations (20) and (21) respectively which were used in the two previous steps are based on the assumption of constant acceleration between two sampling points. Depending on the effective gridding this assumption may be violated. In this case the gridding needs to be refined by inserting new sampling points where needed. This should be repeated until the resulting acceleration error is less than a predefined tolerance. Note that steps 3 and 4 also need to be repeated in that grid refinement loop.

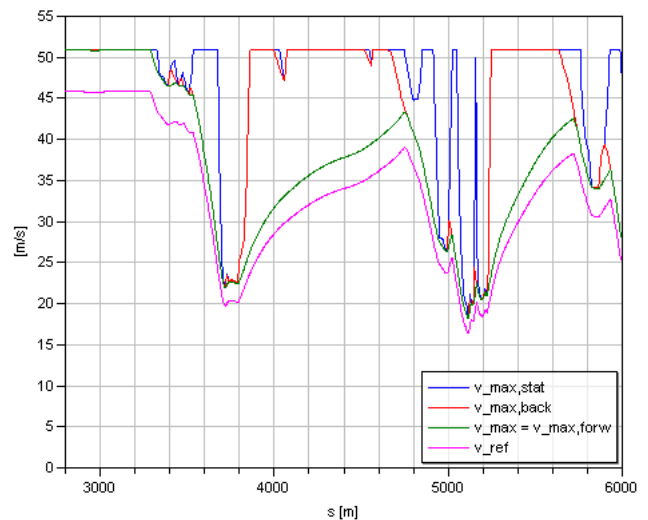


Figure 3: Calculation of the maximal speed profile and definition of a reference speed profile

The decisive maximal speed profile, finally, is

$$v_{\max}(s) = v_{\max, \text{forw}}(s). \quad (22)$$

If the vehicle exactly follows this speed profile then it drives at maximum speed while respecting all physical limits plus considering driver type dependent safety, comfort, and economy relevant margins.

2.6 Reference speed profile definition

The before calculated maximum speed profile can be adopted as a base for the definition of a *reference speed profile* which is suitable for speed control of the vehicle. As an example, linear scaling is applied

$$v_{\text{ref}}(s) = \kappa_v \cdot v_{\max}(s) \quad (23)$$

using a driver type dependent parameter κ_v with $0 \leq \kappa_v \leq 1$ (compare Figure 3 with $\kappa_v = 0.9$).

3 Speed control based on reference speed profiles

3.1 Using acceleration as control variable

The aim of speed feedback control is to make the error between reference speed and actual vehicle speed small. For adjustments of the vehicle speed, accelerations in the interval given by (13) are permitted. Accordingly, a reference longitudinal acceleration a_{ref} is formed by the controller. In a successive module which is not discussed here, the reference acceleration can be transformed into gas and brake pedal positions as accurately as possible e.g. by using nonlinear inverse static or inverse dynamic models [3]. Any speed errors resulting from model inaccuracies or induced by disturbances can be compensated for by the speed feedback control which is described as follows. It turned out that proportional feedback of the speed control error yields satisfactory results, even if the resulting control variable a_{ref} is limited according to \dot{v} in (13). Before being more precise with this issue, prediction of the speed error is introduced.

3.2 Prediction of the speed error

Significant control performance improvement can be achieved by compensation of plant delay. A parameterizable prediction time T_{pred} takes into account summarized lags which may be present in the control loop such as power train or brake dynamics. Hence,

both the reference speed and the vehicle speed are predicted by T_{pred} in advance. The approximation used here assumes that during the prediction time the acceleration remains constant.

The predicted reference speed is simply determined by evaluation of (23)

$$v_{\text{ref}, \text{pred}} = v_{\text{ref}}(s_{\text{pred}}) \quad (24)$$

at the predicted vehicle position

$$s_{\text{pred}} = s_{\text{veh}} + v_{\text{veh}} \cdot T_{\text{pred}} + \frac{1}{2} \dot{v}_{\text{veh}} \cdot T_{\text{pred}}^2. \quad (25)$$

On the other hand, the predicted vehicle speed is

$$v_{\text{pred}} = v_{\text{veh}} + \dot{v}_{\text{veh}} \cdot T_{\text{pred}} \quad (26)$$

The prediction time may also be considered a driver type dependent parameter. If no prediction virtue is wanted then T_{pred} can simply be set to zero.

3.3 Limited proportional feedback

As mentioned before the speed control uses feedback of the predicted speed error

$$a_{\text{ref}, \text{raw}} = \kappa_g \cdot (v_{\text{ref}, \text{pred}} - v_{\text{pred}}). \quad (27)$$

The driver type dependent parameter κ_g is the feedback gain. Finally, the controller must respect the acceleration limits (13):

$$a_{\text{ref}} = \begin{cases} c_{\text{pred}} - d_{\text{pred}} & \text{if } a_{\text{ref}, \text{raw}} < c_{\text{pred}} - d_{\text{pred}} \\ c_{\text{pred}} + e_{\text{pred}} & \text{if } a_{\text{ref}, \text{raw}} > c_{\text{pred}} + e_{\text{pred}} \\ a_{\text{ref}, \text{raw}} & \text{otherwise} \end{cases} \quad (28)$$

Reasonably, all variables take on their values at the predicted position $s = s_{\text{pred}}$. Note that if this kind of limited feedback (28) is used, then the limitation of the speed profile in forward direction is redundant and $v_{\max}(s) = v_{\max, \text{back}}(s)$ should be used rather than (22).

4 Simulation results

4.1 Modelica implementation

Figure 1 shows the total Modelica model we built during implementation and prototype testing of the driver model together with a very simple vehicle model. The speed control given by (27) is implemented in the block *driver*. The block *vehicle* uses the longitudinal acceleration as requested from *driver* as input. It consists of a first order lag element for representation of the power train / brake dynamics. The time constant is set unrealistically high to

$T_{lag} = 1.0$ s to demonstrate clearly the benefit of the speed error prediction concept. Two successive integrators compute speed v_{veh} and position s_{veh} , respectively.

The prime block *speedProfile* needs the tabled road definition as parameter. From that, the maximal speed profile is calculated in multiple steps as explained in section 2.5. A corresponding function is executed at model initialization time and stores the result in a parameter table. Therefore, also driver type and vehicle parameters are needed which are instantiated as records in the total model. During the simulation, the block *speedProfile* provides the predicted quantities needed for (27), (28). To facilitate their calculation all relevant variables are evaluated on the base of the predicted vehicle position (25) and speed (26). The tabled road data and the pre-calculated reference speed profile are correspondingly interpolated.

4.2 Prototypy simulation

This section shows simulation results obtained with the prototypical implementation from Figure 1. The vehicle starts at $s_{veh} = 2900$ m and $v_{veh} = 0$. The *normal driver* record with the parameters given in Table 1 was used.

Table 1: *Normal driver* type parameters

$\mathcal{K}_s = 0.4$	$\mathcal{K}_w = 0.4$	$\mathcal{K}_v = 0.9$	$\mathcal{K}_f = 1.1$
$\mathcal{K}_g = 10$	$\mathcal{K}_p = 0.6$	$T_{pred} = T_{lag} = 1.0$ s	

In Figure 4 the blue line is the pre-calculated reference speed profile (cf. Figure 3). The red line is the simulated predicted reference speed. We find that the simulated vehicle speed (green line) matches very well the reference speed.

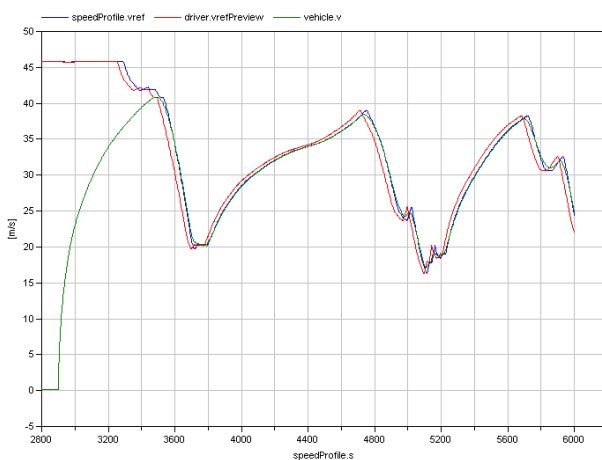


Figure 4: Speed profiles and actual vehicle speed in the simulation

Only at the beginning there is a big gap which is conditioned by the limited acceleration. Figure 5 shows the effective acceleration limits (blue, red) according to (13) and the actual vehicle acceleration (green line). The acceleration potential is fully exploited in the initial phase while there is a large speed error. Later, the driver model keeps some margin from the limits which is due to our choice

$$\mathcal{K}_v = 0.9 \text{ in (23).}$$

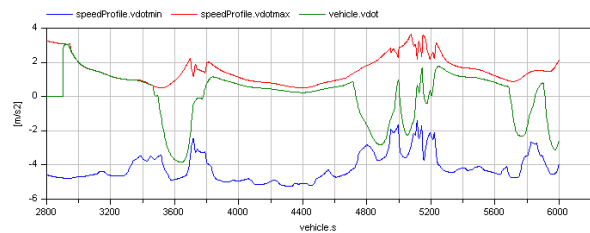


Figure 5: Acceleration limits and actual vehicle acceleration in the simulation

The *driver related degree of utilization of force transmission quota* from (3) is shown in Figure 6 with a blue line. As a consequence of our approach, it must never exceed one. The *physical degree of utilization of force transmission quota* is plotted as a red line. It is obtained by setting $\mathcal{K}_s = 1$ and $\mathcal{K}_w = 1$ in (3) and thus removing the driver type dependent implicit safety margin.



Figure 6: Utilization of force transmission quota in the simulation

5 Experimental results and applicational issues

The Modelica driver model was evaluated at Daimler and found suitable for the purpose of virtual drivetrain endurance tests. Thereupon, the driver model was deployed in a *software in the loop* environment (SiL) in conjunction with a detailed plant model. The functional code in the loop is the control code of an automatic Mercedes-Benz gearbox transmission. The used plant model describes the longitudinal dynamics of a vehicle and has its modeling focus on the 1-D rotational dynamics of the drivetrain.

Figure 7 shows a top-level screenshot of the model [6]. The calculation of the reference speed profile (23) as described in section 2 serves as reference speed and was integrated in the overall car model (Figure 7) which in turn was exported as a DLL for the SiL environment. For the code export we used the C-Code generated by Dymola 6.2 wrapped with an API for the co-simulation tool BACKBONE, a proprietary Daimler program.

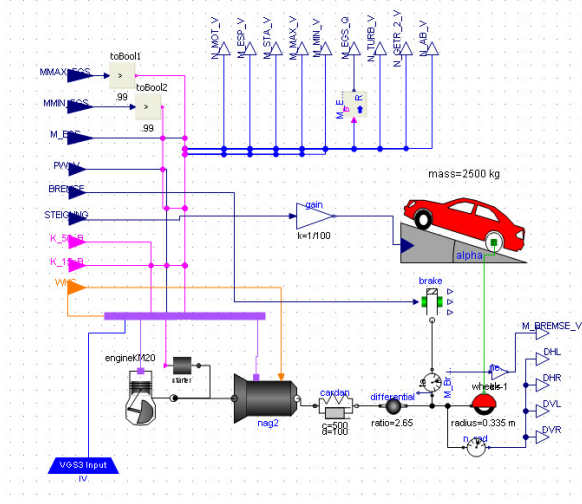


Figure 7: Modelica car model for SiL export

For SiL control of this model at Daimler an enhanced feedback control was used rather than (27). It also accomplishes the assignment to both throttle and brake pedal based on PI-control of the vehicle speed error. The driver specific parameters used for the calculation of the speed profile turned out to be useful for the calibration of the total SiL driver model.

This specific deployment of the SiL is used for virtual endurance testing of the drivetrain. The tracks we use are the same that our testing teams drive in reality. With the virtualisation we are able to

- examine the impact of code updates on the endurance of the hardware (gearbox and drivetrain components),
- detect bugs in the code, and
- calculate load collectives.

All this can be done in a fast and absolutely low cost manner. So far, at Daimler, simulation of load collectives for gearboxes primarily had taken place with special software which, however, didn't include the functional code. SiL simulations of this kind had been done by using a fixed speed profile derived from experimental measurements or a load collective simulation. The reference speed input to the SiL was therefore car specific and could not be used for other vehicle configurations. Moreover, the reference

speed had been time scheduled rather than position scheduled. Hence, a cumulative error in the calculation was unavoidable due to the deviation between desired and actual speed: After some simulation time on long tracks (some 100 km) the vehicle's position did not match the position the reference speed was assigned to. As a result of this error, peculiar situations occurred in the simulation such as full throttle while downhill driving etc.

With the new method of car specific speed profile calculation coming along with position dependent driver action we are now able to use the SiL directly for load collectives simulation without the need for extra software. Only the topology of a track is needed and track specific restrictions, such as speed limits, obligatory stops etc.).

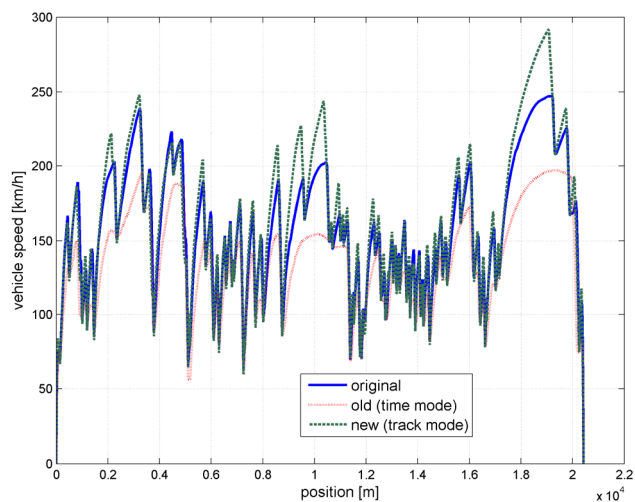


Figure 8: Comparison of simulation results (speed over vehicle position)

For illustration of the realized progress Figure 8 shows a comparison of speed profiles. The first curve (dep. "original") is produced by our special program for load collectives calculation, which used to be the input for SiL simulations. This speed profile is considered a benchmark for the new method. Two SiL simulations were made, one with the previously used method (dep. "old (time mode)") and one using the new approach (dep. "new (track mode)"), each with a similarly configured (engine power, mass) car model on the same drive track.

With the new method, the resulting vehicle speed fits the benchmark speed significantly better than with the old method despite only road track data but no direct information of the benchmark speed was processed. This also applies to the primarily relevant criterion for drivetrain endurance i.e. the load collective. In Figure 9 - Figure 11 one can see the compari-

son of the three variants w.r.t. the load shapes at the cardan shaft (torque over engine speed).

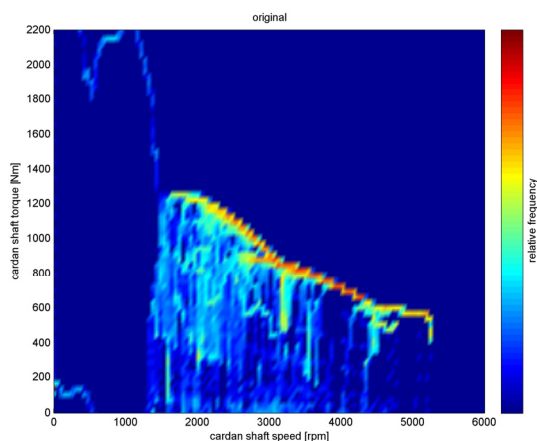


Figure 9: Simulated cardan shaft load collectives (torque over speed), original version.

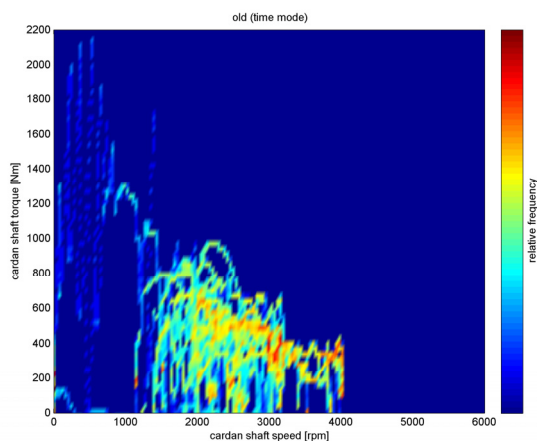


Figure 10: Simulated cardan shaft load collectives (torque over speed), old version using time mode.

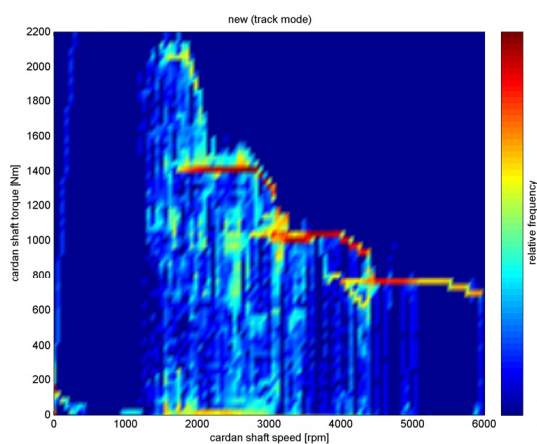


Figure 11: Simulated cardan shaft load collectives (torque over speed), new version using track mode.

It is obvious that the distribution yielded with the new method matches better the benchmark than the old approach. Analogous results are obtained for all drive tracks that Daimler uses in the gearbox development. The flexible car dependent parameterization obviously yields better robustness of the load collectives results.

With the new method for the calculation of time independent, vehicle specific speed profiles we made an important step towards the evaluation of load collectives in conjunction with SiL simulation. Our input to the simulation from now on consists in time independent track data and is identical for all vehicle models, regardless of car weight or installed engine power.

6 Conclusions

A new method for quick automatic calculation of reference speed profiles which are applicable for automatic gearbox testing was developed at DLR and implemented using Modelica. The resulting speed profiles are specific for the assumed vehicle data. Moreover, they can easily be adapted by tuning of various parameters which are interpretable to represent different driver behavior.

At Daimler, the algorithm is now used for both flexible and reproducible generation of load collectives for virtual drivetrain endurance testing. The new approach replaces the less efficient procedure where static time-dependent speed profiles were taken as inputs which had been produced from special load collective generation software or driving experiments separately for each car type.

7 Acknowledgement

The presented results were compiled in the context of the ITEA2 project Modelisar [4].

References

- [1] Aguilera, V., Glaser, S., Arnim, A.: An advanced driver speed assistance in curves: risk function, cooperation modes, system architecture and experimental validation. Proc. IEEE Intelligent Vehicles Symposium. Las Vegas, Nevada, 2005,
- [2] Jimenez, F., Aparicio, F., Paez, J.: Evaluation of in-vehicle dynamic speed assistance in Spain: algorithm and driver behaviour. IET

- Intelligent Transport Systems, Vol. 2, No. 2, 2008.
- [3] Thümmel M. et. al. Nonlinear Inverse Models for Control. Proc. 4th Int. Modelica Conf., Hamburg, 2005.
 - [4] “*MODELISAR Project Profile*” 2008, http://www.itea2.org/public/project_leaflets/MODELISAR_profile_oct-08.pdf
 - [5] Schlabe, D., Knostmann, T., Bunte, T.: Scade Suite Modelica Interface. Proc. 8th Int. Modelica Conf., Dresden, Germany, 2011.
 - [6] Chrisofakis, E., Junghanns, A., Kehrer, C., Rink, A.: Simulation-based development of automotive control software with Modelica. Proc. 8th Int. Modelica Conf., Dresden, Germany, 2011.



Optimization-Tool for local renewable energy usage in the connected system: “Building-eMobility”

Dipl.-Ing. Torsten Schwan
 Prof. Dr.-Ing Bernard Bäker
 Dresden University of Technology
 Institute of Automotive Technologies Dresden
 George-Bähr-Str. 1c, 01062 Dresden
 schwan@iad.tu-dresden.de

Dipl.-Ing. René Unger
 Dr.-Ing. Beate Mikoleit
 EA EnergieArchitektur GmbH

Königsstr. 2, 01097 Dresden
 rene.unger@ea-gmbh.de

Dipl.-Ing. Christian Kehrer
 ITI GmbH
 Webergasse 1, 01067 Dresden
 Kehrer@iti.de

Abstract

Renewable energy production and decentralized energy storage as well as optimized usage of existing energy resources are matters of rapidly growing importance. Even today in building architecture as well as modern mobility concepts these technologies are major cost drivers.

Staying abreast to these changes, EA EnergieArchitektur GmbH together with IAD TU Dresden are developing a simulation tool to identify and optimize the potentials for building specific energy storage and production as well as optimized usage strategies on the consumer side.

Furthermore the simulation tool allows analyzing the smart integration of new eMobility concepts. In this it works as a test bench for system wide energy management with priority on charging strategies for such vehicles from the decentralized power supply.

Keywords: renewable energy; eMobility; modeling

1 Why a holistic energy simulation for car and building

Today, there are various technologies available to provide local renewable energy, for example: micro-wind-turbines, photovoltaics, solar heat, heat pumps and combined heat and power units (CHP).

These energy systems use direct natural energy resources like wind and sun or renewable fuels like wood, bio-gas or even vegetable oil. The availability and efficiency of these resources differ greatly depending on the specific location. Furthermore these energy systems are expensive in money and production resources. Therefore it is important to find an optimized configuration before installing an energy system in a specific building.

A second thought on optimizing renewable energy includes the time of availability. There is no sunshine at night. Is it better to store the daylight energy using batteries, charge a heat storage or to install the photovoltaics facing westwards thus providing more energy when demand is high - in the evening?

In the course of the increasing demand on electric mobility, the need for charging concepts has risen. Both power and energy have to be provided. Synchronization of the demand on energy and storage as well as on availability in a building is an important fact for future energy-management systems.

Electric Mobility as mobile storage with constraints to availability (docked, undocked) and requirements from lifestyle (e.g. 100% charge in the morning) adds further complexity to the system.

The energy system layout for a specific combination Building-eMobility as well as the research and development of optimized energy-management algorithms are two engineering tasks which demand for a

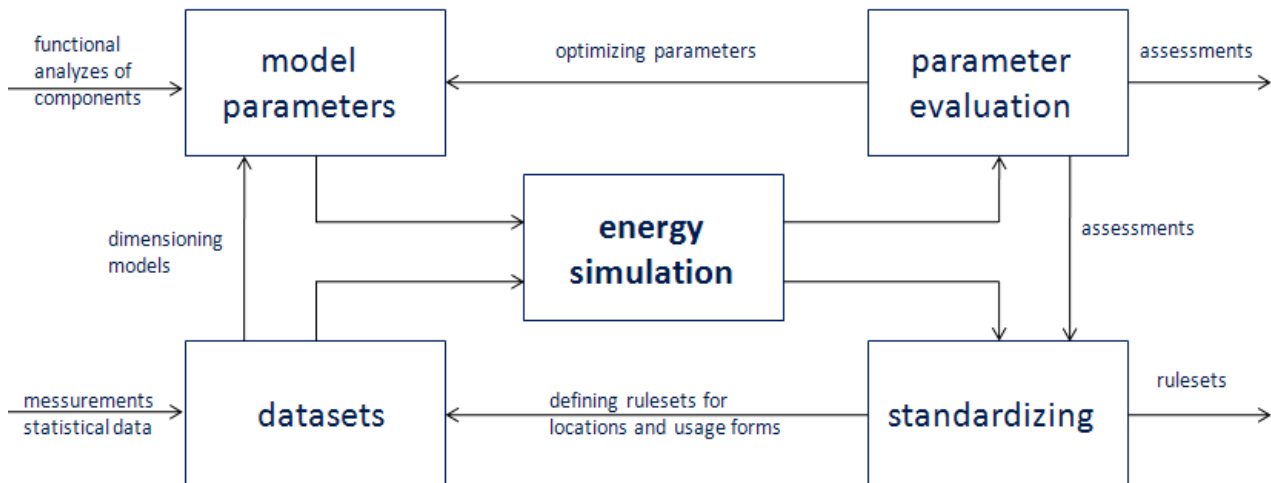


Fig. 1: Structure of the energy simulation process

dynamic simulation covering all macroscopic aspects of the system.

Such a simulation system is not available, yet. Existing Software either covers only one of the subsystems (e.g. PVSol) or it does not include costs and dynamics using precalculated balances instead.

This paper shows a different approach using the interdisciplinary modeling Language Modelica to implement physics and information flow as well as cost-specific behavior of building-related renewable energy systems in the same set of differential and algebraic equations. Additionally the object oriented modeling concept allows to describe well-arranged and understandably system behavior of the power generation, storage and consumption components.

The tool under development presented in this paper is able to simulate and evaluate the energy flow in the holistic connected system “Building-eMobility”. It will be used to layout the most energy and cost efficient combination of components as well as for testing intelligent energy-management algorithms in much-faster-than-real-time (accelerated SiL).

Since the main focus is on system layout and benchmarking the location specific energy and cost efficiency, the applicable level of detail is an important fact on model design.

2 Concept of the energy simulation model

The basic conceptual design requirements of the simulation model can be deduced from the desired simulation results.

The development of precise rules for the renewable energy system layout, depending on local climate and planned usage, requires a reduced set of possible subsystems for the later system composition (i.e. micro-wind-turbines, electric car, electric bike, etc.).

Additionally a common set of input-parameters based on availability (i.e. weather statics) and building design (heat-transmission, inhabitants) needs to be defined.

The analysis of a specific location over long time periods (>1 year, <1 minute step) with regard to different system layouts, cost, availability and energy independence requires easily replaceable subcomponents (i.e. Photovoltaics / Solar Heat) with integrated cost functions as well as the use of “real-world” measurements as input-data.

To enable parameter variation the calculation needs to be fast, thus phenomenological descriptions instead of real physics should be used wherever possible.

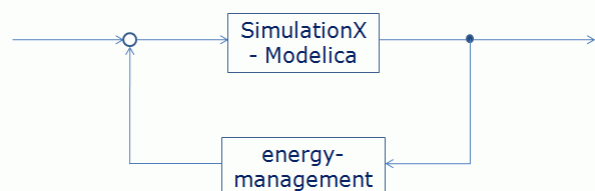


Fig. 2: SiL-application of the simulation environment including energy-management-algorithms

The finished simulation system will act as a test bench for advanced energy management algorithms. Therefore the integration effort for a simulated or SiL-energy-manager needs to be reasonable. In these

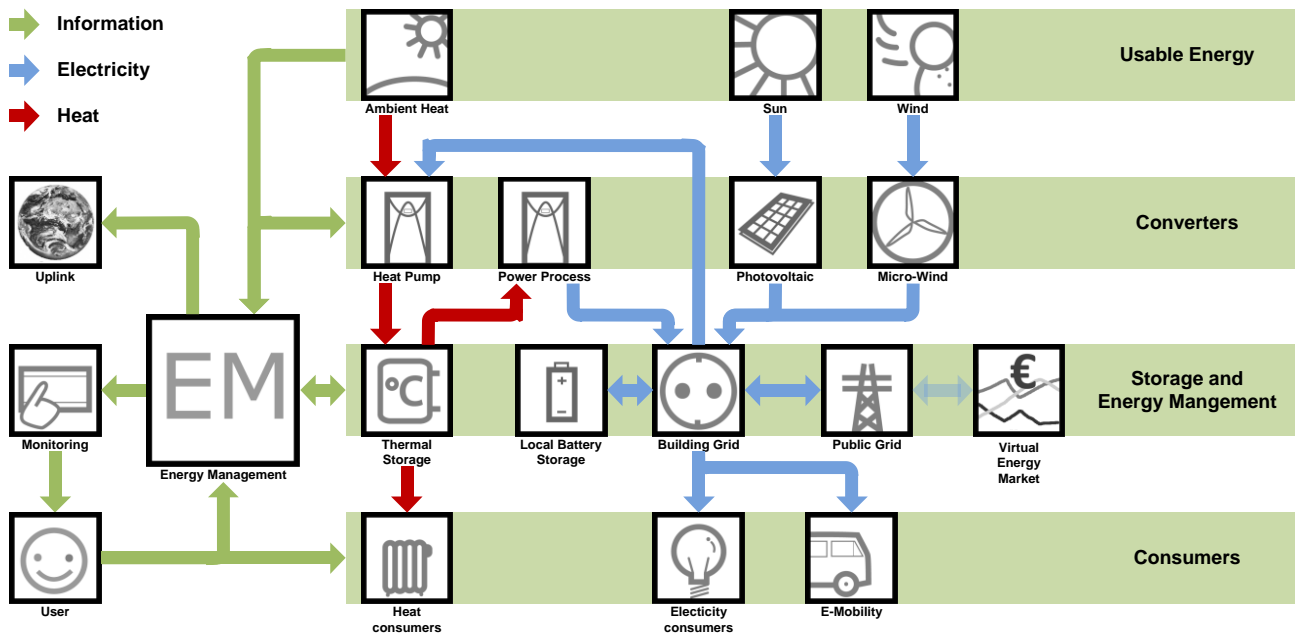


Fig. 3: Energy and information flow in a specific scenario building - vehicle

tests critical system states (e.g. power shortage, empty car battery in the morning) should be detected automatically and calculated in higher time resolution. Fig. 1 shows the basic simulation process.

The SiL-energy-manager could be implemented as an algebraic equation system including all relevant measurement and control signals. An integration of an outstanding control-block as energy-manager implemented in another environment like Matlab/Simulink is also imaginary.

The resulting simulation consists of a library of subsystem-models for each specified component. These sub-models are connected using an energy-bus and a cost-function-bus. Especially interesting components like micro-wind-turbines and batteries are available as fast phenomenological models as well as exact physical models. Besides the energy-components, different models for inhabitants, climate and utility company were created.

3 Examples for Subsystem-Models

In this paragraph an abstract of the implemented Subsystem-Models used in the energy simulation tool is described. Special attention is turned on the model requirements dealing with the discrepancy between fast system calculation time and precise simulation results. For implementation partially existing approaches were used.

3.1 Micro-Wind-Turbine

The Micro-Wind-Turbine model calculates the electrical energy output of a building integrated wind power plant. Input-data is wind speed and direction for a specific location, based on “real-world” measurements or statistics. With attention to all existing physical relations like angle of incidence, acceptable operating range and aerodynamic configuration of the turbine, the wind power absorbed by the turbine during the integration time step is calculated.

Depending on the specification of the gearing mechanism and the designated type of electric generator, i.e. asynchronous machine, the model simulates the generated electrical power from the turbine power.

To improve calculation time and to avoid problems with the internal logic of modern smart inverters the phenomenological behavior is replicated. Therefore the model is based on generator specific characteristic curves (e.g. dependency of open-circuit-voltage and generator torque on generator velocity) in a closed loop control with turbine power as reference instead of magnetic coupling and switching diodes.

The behavior of the net-coupling inverter is rendered in a similar way with additional inputs for external power management.

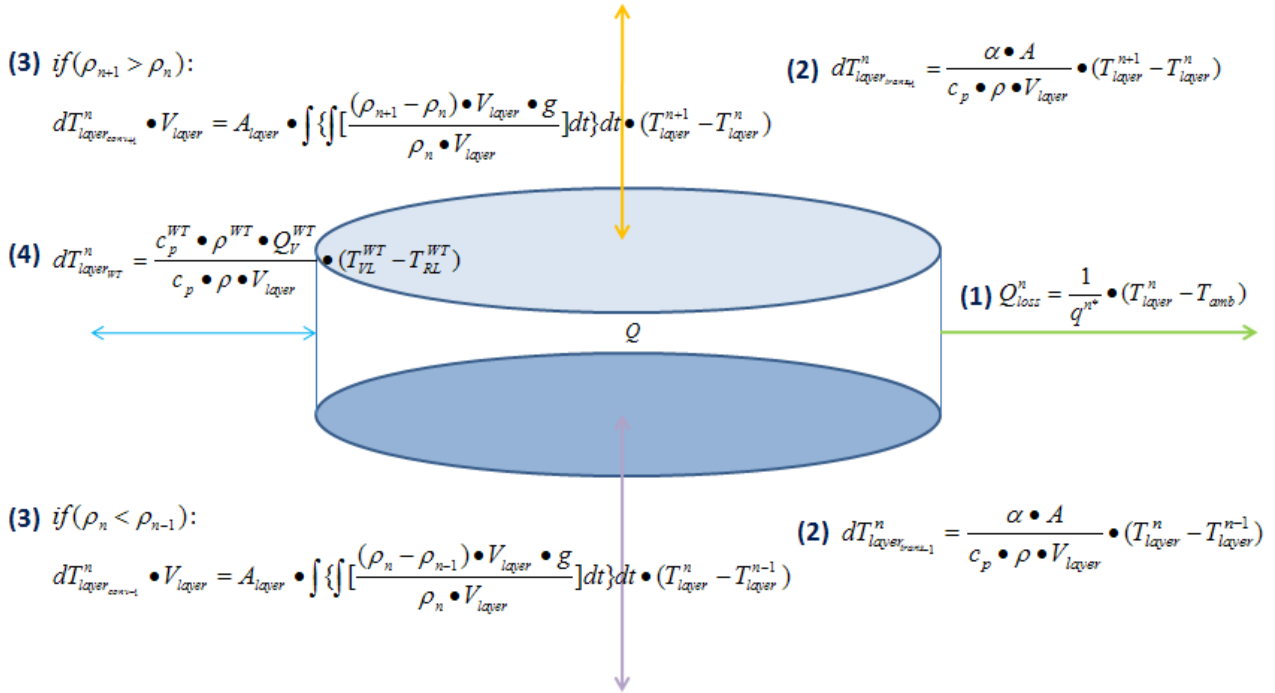


Fig. 4: Differential equations describing the behavior of the heat storage tank

3.2 Heat Storage

The Heat Storage model describes mathematically the behavior of the temperature spreading in a tank which is used to save thermal energy. The model considers the energy losses of the tank through the isolation (1) and the heat transfer (2) as well as the heat convection (3) between the different temperature layers in the tank. The describing differential equations for one layer are shown in fig. 2. Also the heat dissipation and supply of a layer (e.g. by heater or Heat Pump) is implemented in the model (4).

The model will be parameterized by the physical dimensions of the tank and the integrated heat exchanger. Another describing parameter set consists of the thermal characteristics of thermal storage media. Output of the simulation model is the amount of energy put in or dissipated from the Heat Storage and the temperature spreading in the tank.

Although there have been existing a lot of different models for Heat Storage systems the presented model have become necessary because these models were so detailed that the simulation time would have been very long. So the presented model was adapted that way that only the behavior of the temperature spreading in the Heat Storage will be contemplated.

3.3 Battery

Electrical Batteries are a major field of research at the IAD TU Dresden. Based on these long-term studies, Batteries are described mathematically to render the exact electrical behavior of a black-box at its terminals. The fast model defines a number of battery strings, each consisting of a specified number of in series connected battery cells.

These cells are parameterized by measured characteristics (e.g. impedance, open-circuit-voltage) of different types of battery cells (e.g. Li-Ion). Out of these characteristics the model calculates the realizable power supply of the battery depending on the state of charge and other characteristic battery conditions (e.g. cell temperature).

This battery model allows simulating the power supply of different types and dimensions of batteries within a very short simulation time because inner chemical processes were neglected. Additionally it enables the calculation of the specific lifecycle-cost including usage dependent cell aging.

3.4 Heat Pump

A heat pump is a machine to transform heat from a reservoir with lower temperature (e.g. ambience) to a higher temperature level for heating and storage. The focus is put on electrically driven pumps.

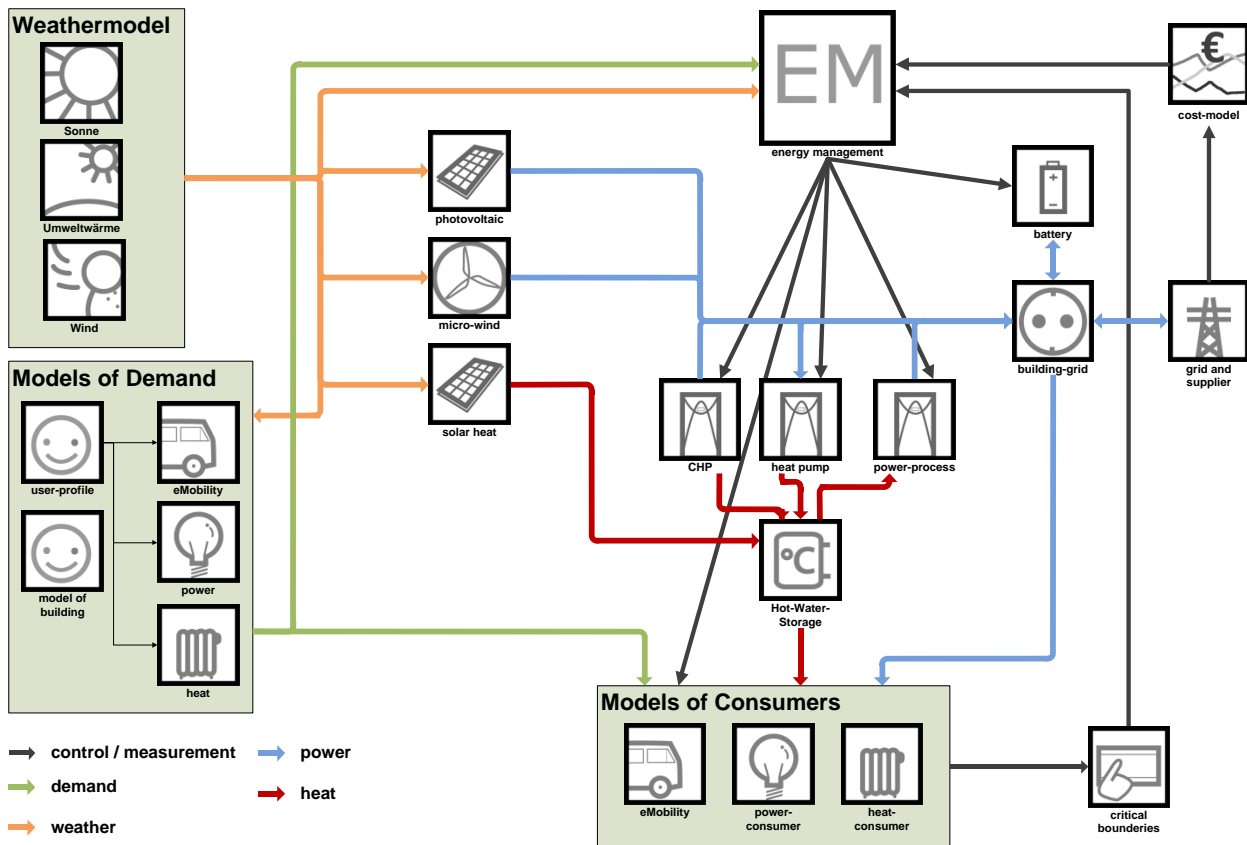


Fig. 5: Simulation concept and subsystem models for weather, demand, resulting consumers and energy generation, storage and management subsystems

In the process the working medium (e.g. CO₂, R407C) is subjected to a thermodynamic cycle. Although it is possible to implement this thermodynamic cycle in a mathematical model, the model level of detail for exact representation would be far too complex for the short simulation time requirement. Therefore another approach was chosen for modeling this system comparable to other subsystem-models.

The method used for modeling the Heat Pump divides two phases for the calculation of the supplied heat of a Heat Pump, the dissipated heat flow between the working medium and the heating medium and the rise of the enthalpy of the heating medium per time unit. These two heating flows are equal in steady states. So for the simulation of the heating flow of characterized Heat Pumps the model uses manufacturer data for the input-data-related steady state heat and electrical power flow.

The dynamic of the heat pump respective input-data and heat flow modification as well as switching operations necessitates other modeling methods. The problem is solved on the one hand by model partitioning between heat dissipation and generation.

On the other hand the different types of switching operations (normal switching, de-icing) are modeled

as loop controller with different time constants.

Fig. 6 shows the model structure used to calculate the switching transient behavior of a Heat Pump after a de-icing process because of coincidental switching of ventilator, circulating pump and heat pump. Also simulation results for behavior of heat power at normal switching events compared with after-de-icing events are presented.

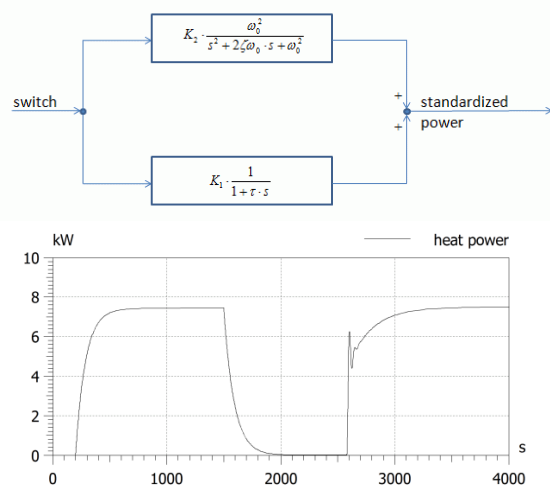


Fig. 6: Model of switching transient characteristic after de-icing (above), simulation results (below)



Fig. 7: Simulation library pictured as mind map

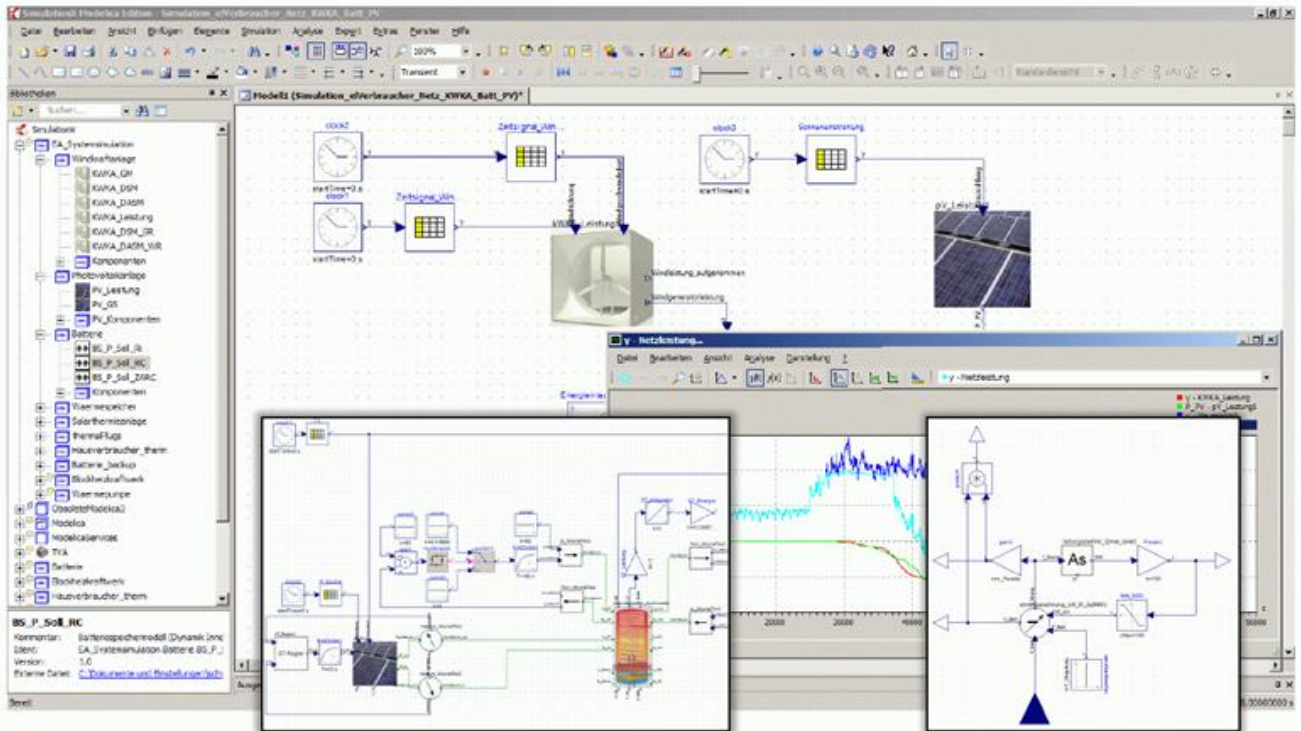


Fig. 8: Screenshot of the simulation model including the used libraries

4 Gathering Input-Data

Simulation results are only as good as the input-data they are based on. Therefore the important groups of data like weather and climate, technical characteristics of components and energy statistics have to be gathered for specified simulation scenarios.

Weather and climate are important for the energy output as well as the demand. Statistics for local climate are available from various sources like “Deutscher Wetterdienst” but these are often cost intensive. To analyze weather dynamics we use long term measurements (i.e. wind speed) with high time resolution (1s) at the specific location. Since these datasets are too huge for reasonable use within SimulationX, extensive preprocessing is done. The resulting parameters are used in a weather sub-model to create stochastic signals with characteristics similar to the original measurement.

The technical characteristics mostly depend on the datasheets of the used components. Special emphasize is put on the characteristics of micro-wind-turbines (a primary technology of EA GmbH) and battery storage (IAD).

Energy-data is generated similar to weather data based on energy-suppliers statistics, market-data and direct measurements at our research sites. The result-

ing weather, market and usage model can provide stochastic output signals based on season and time of day.

5 Synthesis of a holistic model

Based on the simulation scenario the holistic model consists of the corresponding parameterized submodels for the energy system components (i.e. 10m² PV, heat-pump, 5 kWh battery, 2 cars, etc.). Respectively the models for users, environment / weather, consumption and provider are added and configured. The according “real-world” input-data for the scenario is stored in external files and fetched at simulation time.

To simplify the simulation model and to improve the clearness, signals like velocity and direction of wind, are unified to bus systems. The main busses connecting the subsystem-models are energy-bus, cost-bus, environment-bus and energy-management-bus. Due to the calculation time requirement, most connections are signal couplings (only variable information) as opposed to physical coupling (real physical variables).

These couplings are divided into special-defined connectors for thermal and electrical elements. Therefore these connectors allow the connection of

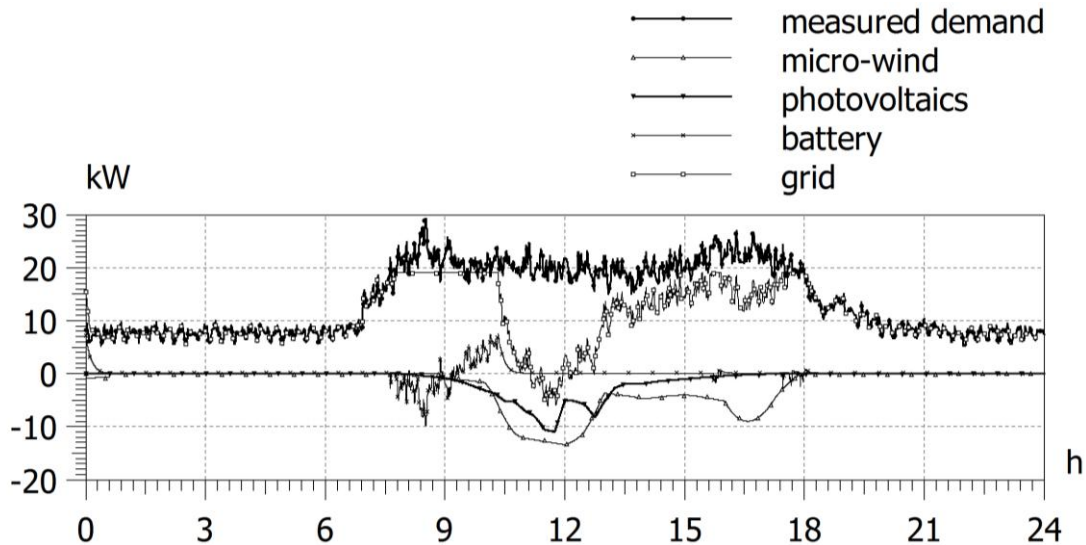


Fig. 9: Results for simulated electrical power

different model-types which are interrelated. Realizing the interrelated behavior of these subsystem-models the connectors transfer all power-relevant data.

6 Examples for simulation results

The following example shows the simulated behavior of the thermal and electrical energy flow in an office building in central Germany. The input data is based on measurements from March 2010.

In the simulation run the building is outfitted with five Micro-Wind-Turbines (3m housing, 3.5 kW rated output), photovoltaics rated at 16.5 kWp and a li-ion battery system of 10 kW rated power and 4.32 kWh storage).

Besides the configuration of the energy producing and storing systems in the simulated building the used energy-management-algorithm for system controlling is essential for the calculated power behavior. So in the presented simulation was defined that the building-integrated battery will be discharged above a specific electrical power demand and charged otherwise preferred by local renewable produces energy. The battery application primary use is peak power reduction.

Fig. 9 shows the simulated electrical power. With the above configuration, the peak power drawn from the grid can be significantly reduced by 10 kW. Almost all renewable energy is used locally. The peak at noon could be covered with a marginally bigger battery or used for heating. In case of the selected con-

figuration and parameter-set the cumulative electrical energy demand could be reduced about 20%.

The simulated thermal subsystem contains an oversized 239 m² flat plate thermal collector and a 3.5 m³ hot water storage tank. In the application the solar heat collector charge the heat storage tank. That configuration decouples the heating system in the building from heat sources. In the simulation, all conventional heating is combined as “grid”.

The combined heating system was controlled by the storage temperature at a specific layer as reference value. The heat was then extracted from the tank on demand, observing tank layer and temperature spread.

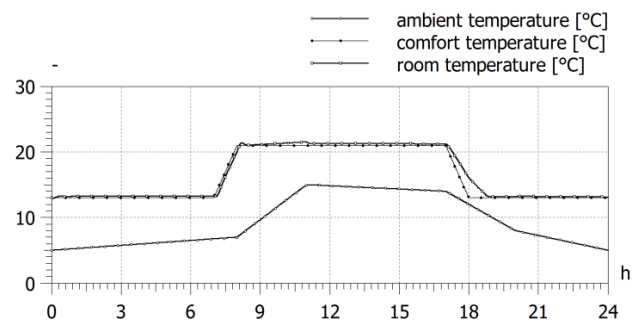


Fig. 10: Simulated temperature characteristics

Fig. 10 shows the initiated and simulated characteristics for ambient, comfort and room temperature. The comfort temperature refers to nominal temperature characteristics in office buildings. Thereby the room temperature has to be reduced in the night in order to save energy.

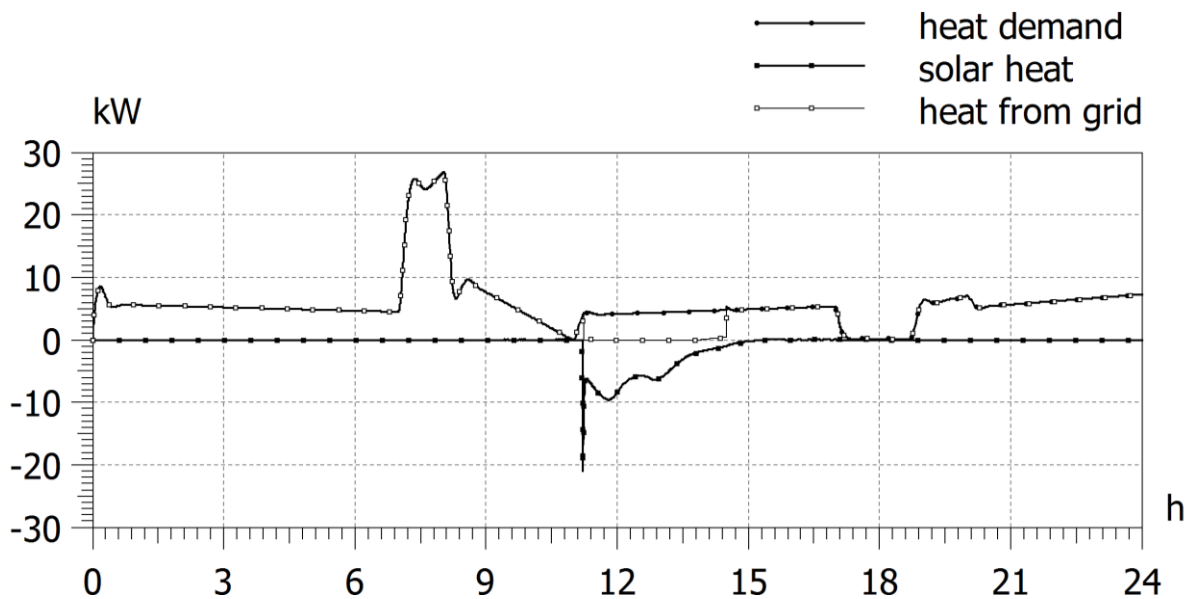


Fig. 11: Results for simulated thermal power

Fig. 11 shows the peak thermal energy demand in the morning (switch to daytime temperature), important for the system layout. A second aspect is shown in the solar heat graph. Only around noon, the collectors get hot enough to charge the storage tank. Additionally the decoupling of solar heat and heating cycle can be seen. The tank size in this case is enough for the day cycle.

In case of such an oversized configuration and parameter-set the cumulative heat demand could be reduced only about 8%. Although it is a sunny winter day, the collectors though oversized, cover only a small part of the heating requirements. Lower temperature heating systems or direct use of the solar heat would be options to find an applicable layout.

Another important result of the simulation addresses the consideration of the influence of the heat produced by the people working in the building. The simulated demand of the building differs about 18% between calculation runs with and without the standardized number of office workers in the building.

Finally the defined requirement on “faster-than-real-time”-simulation could be achieved including sufficient detailed results. The realized factor between the simulation time and the simulated time is 1:1000.

7 Conclusion and future developments

As for today, it is possible to simulate the energy flow in a complete combination eVehicle-Building, given a specific configuration of the energy system. Based on the results the configuration can be optimized manually and validated afterwards. Energy-management algorithms can be tested within the simulation. Energy usage and wastage are analyzable and comparable. Including modern charging concepts for eVehicles which are dedicated to a simulated building is also possible.

The future development aims to extend a database with simulation results and input-datasets, including different combinations of buildings, vehicles, locations and usages. This database will also be connected to acknowledged tools for detailed component layout (i.e. PV calculation, heat demand). Furthermore the process of parameter variation and optimization for parameters like energy generation, usage, lifecycle cost and independence shall be automated.

Long term objectives are an independent application and standards for assessment of local renewable energy systems.



Fig. 12: Energy Monitoring (left side); reference building for modular renewable energy management system (right side)

8 Research project: “Residence and Mobility”

The described tool is developed within the research project “Residence and Mobility”. The aim is to cover all energy demands of a family and their individual lifestyle with the renewable energy provided around the building they live in. The research project is encouraged with subsidies from the European Union and the Sächsische Aufbaubank (SAB).

Fig. 12 shows the reference building implementing the new technologies including in-house micro-wind-turbines, photovoltaics, CO₂-heat-pump and 12m³ heat storage tanks. A user friendly monitoring system shows the workings of the energy system

References

- [1] H. Winkelmeier: Energiesysteme - Windenergie - 02 - Bauformen und Aerodynamik von Windkraftanlagen. energiewerkstatt^o, 2005.
- [2] P. Fritzon: Principles of Object-Oriented Modeling and Simulation with Modelica 2.1. Wiley-IEEE Press, 2003.
- [3] R. Unger; T. Schwan; B. Bäker; B. Mikoleit: Optimization tool for local renewable energy usage; ITI Symposium 2010.
- [4] S. Kutter; R. Falsett; B. Bäker; L. Morawietz: Dynamische Modellierung des makroskopischen, thermoelektrischen Verhaltens von Lithium-Ionen-Energiespeichern. Tagung: Elektrik/Elektronik in Hybrid- und Elektrofahrzeugen, Haus der Technik, München, 17./18. November, 2008.
- [5] T. Schwan: Aufbau eines 12V-Lithium-Ionen Batteriedemonstrator, Technische Universität Dresden, IAD Diplomarbeit, Fahrzeugmechatronik, 2009.
- [6] SimulationX-Hompage der ITI GmbH. <http://www.simulationx.com>
- [7] G. Reiner; E. Shafai; R. Wimmer; D. Zogg; H.R. Gabathuler; H. Mayer; H.U. Bruderer: Kurztestmethode für Wärmepumpenanlagen, ETH-Zürich, Sulzer Fritherm AG, 1998.

Development of a Modelica Library for Simulation of Diffractive Optomechatronic Systems

Thomas Kaden

Klaus Janschek

Institute of Automation, Faculty of Electrical Engineering

Technische Universität Dresden, 10162 Dresden

Thomas.Kaden@tu-dresden.de

Klaus.Janschek@tu-dresden.de

Abstract

The proper operation and performance of optomechatronic systems is fundamentally affected by changes of the relative geometry caused by thermal influences, mechanical displacements and vibrations. Such extrinsic and intrinsic disturbances can be compensated by active control of optical elements like lenses, diffraction gratings or laser sources. In the context with system design and performance analysis tasks it is big challenge to model and simulate the coupled optomechatronic behavior including closed-loop control and disturbances properly on a representative level.

A promising approach is the integration of diffractive optic models in the well established physical object oriented modeling environment Modelica[®], which offers already a broad support of multi-domain libraries, e.g. electrical, mechanical and thermal.

Therefore the basic modeling requirements for diffractive optical elements are outlined followed by a discussion of possible problems and solutions for a computationally efficient implementation of a two-dimensional spatial *optical library* for Modelica-based simulation environments.

Keywords: Modelica; Diffractive Optics; Optical library

1 Introduction

The application of optomechatronic systems is increasing constantly. Examples are telescopes with adaptive optics [4], motion compensated cameras [7], [8], diffraction based sensors for on-line textile inspection [1], optical Fourier processors and correlators [9] or interferometer arrangements.

Diffractive optical components are used in optomechatronic systems mainly for the purpose of fast on-line signal processing.

However such systems are affected by environmental influences. Mechanical displacements, vibrations and thermal distortions can significantly affect the proper function and accuracy of the optical subsystem due to changes of the optical path length. Therefore in the context with system design and performance analysis tasks the coupled optomechatronic behavior including closed-loop control and disturbances must be modeled and simulated.

In this paper the integration of diffractive optics models into the physically object oriented modeling environment Modelica is discussed. The requirements for physical object oriented optical models with diffractive optical functionality and interfaces are outlined. In particular the feasibility of two-dimensional spatial interfaces in Modelica is further evaluated and a tool-independent concept for an *optical library* is proposed. The use of Modelica's external object / function interface is motivated through the insufficient support of large matrices in existing Modelica tools.

2 Simulation of optomechatronic systems

Optomechatronic systems are usually composed of an arrangement of electronic, mechanic and optical elements [6] as shown in Fig. 1. Especially the optical functionality depends on a fixed arrangement of optical elements providing exact optical path geometry.

However the operation of mechanical and electrical components is corrupted by geometrical or thermal intrinsic disturbances of the arrangement of optical components. Additional errors are inserted by extrinsic disturbances like shock, vibration or environmental temperature changes.

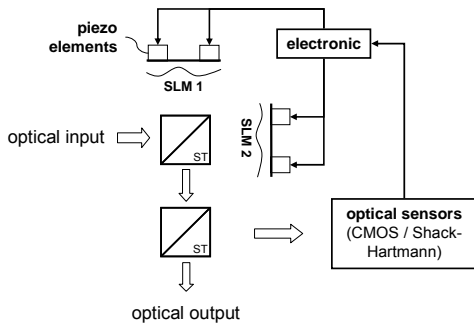


Fig. 1 Optomechatronic System

The state of the art computer based physical object based modeling and simulation environments focus on two main areas:

- Simulation tools focused on optics like Ze-max [17] including detailed geometrical and diffractive optics modeling, but without or only with insufficient mechatronic functionality,
- Multi-domain object oriented tools, e.g. Modelica-based, incorporating a broad mechatronic functionality, but up-to-date without optics functionality, in particular without any diffractive optics [16].

In recent years the object oriented, equation based modeling language MODELICA established mainly focusing at the modeling and simulation of electrical, mechanical, chemical and thermal systems.

The usual approach is to decompose a physical system down to single model components. The entities of components form a package and all packages belonging to one physical domain create a library.

Modelica currently supports electrical, mechanical, multibody mechanic or thermal libraries. Up to now there is no library in Modelica for simulating diffractive optical elements as needed for the simulation of optomechatronic systems, (Fig. 2).

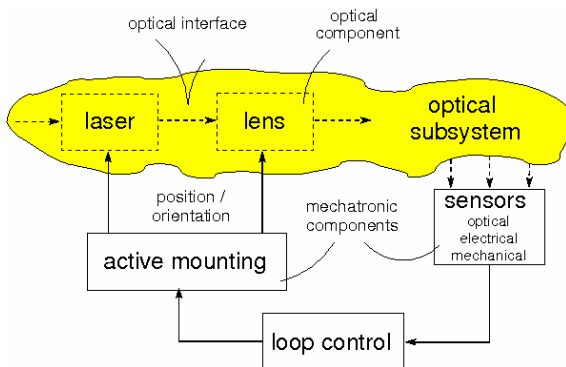


Fig. 2 Optomechatronic system model

The basic problem is the missing support of Modelica for spatially distributed variables and equations [2].

Nevertheless the following analysis of one of the most important diffraction equation shows that basically matrix computation will be needed to implement a diffractive optical functionality.

3 Scalar wave optic

3.1 Rayleigh-Sommerfeld diffraction integral

Many optical phenomena like diffraction or interference of monochromatic coherent light waves can be described in a beneficial manner by scalar wave theory, which leads to a simplified formulation for diffraction phenomena. According to the HUYGENS-FRESNEL principle the complex amplitude distribution of light at a single point behind an aperture can be described by a weighted sum of spherical waves originating from every point within the aperture [3]. This principle can be stated in mathematical form as the well-known RAYLEIGH-SOMMERFELD diffraction integral (1st solution) given by [3], [11]:

$$A(\vec{r}_1) = \iint_{\Sigma} E(\vec{r}_0) \frac{e^{(j \cdot k \cdot r_{01})}}{r_{01}} \cdot \cos(\vec{n}, \vec{r}_{01}) \cdot ds \quad (1)$$

Note that the diffracting aperture is assumed to be planar. For a statically geometric configuration, the integral (1) is independent of time and it can be characterized as algebraic relation between input and output.

3.2 Numerical implementation

As there is usually no analytical solution for (1) the spatial integration must be solved using numerical two-dimensional approximations on digital computers. The two-dimensional spatial discretization leads to a $[N \cdot M]$ matrix representation for a given aperture plane, (Fig. 3).

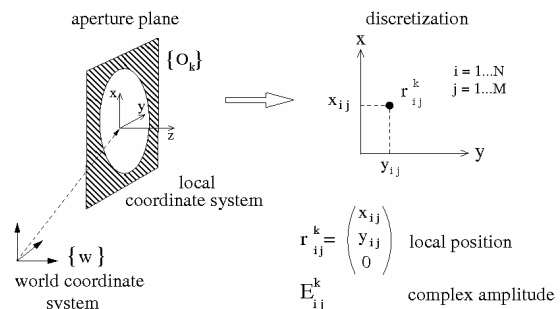


Fig. 3 Discretization of aperture plane

Integration of (1) now has to be replaced by $N \cdot M$ complex multiplications and summations for every output point. Therefore the calculation of an output plane of equal size requires $(N \cdot M)^2$ complex operations. For higher optical resolutions (e.g. $N=M=1024$ pixel) the point wise solution of (1) is a very time consuming task.

3.3 Angular spectrum Method

An optical system can be generally considered as a two-dimensional spatial linear system. Equation (1) can then be solved using Fourier methods, in particular by application of the convolution theorem, assuming spatial invariance (parallel planes). Thus the computational effort for solving of integral equation (1) can be significantly reduced by application of standard fast Fourier transform (FFT) algorithms, complex matrix multiplication with a propagation kernel $H(f_x, f_y)$ and inverse fast Fourier transform (iFFT), (Fig. 4).

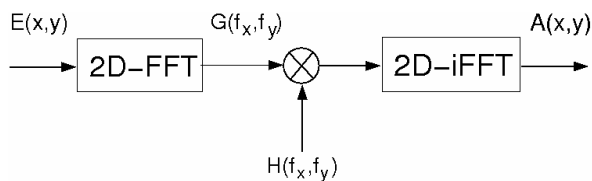


Fig. 4 Principle of angular spectrum method

The propagation kernel is given by [3]:

$$H(f_x, f_y) = e^{j \cdot k \cdot z \cdot \sqrt{1 - \lambda^2 \cdot (f_x^2 + f_y^2)}} \quad (2)$$

This method is orders of magnitude faster than the point-wise calculation of (1).

Assuming the more general case of non parallel planes, this Fourier based method can be used as well, but frequency mapping must be implemented to compensate for the plane rotation [12], [13]. However this step involves *interpolation* due to a restriction inherent to the FFT, namely the fixed sampling of the frequency domain.

4 Concept for an optical library in Modelica

Object oriented modeling of a physical system or component in general needs to include two basic aspects [2]:

- The *interface* of the optical component to pass and access data.

- The *optical component* and its internal encapsulated functionality.

The main properties of optical elements are discussed from the point of view of object oriented modeling as well as there integration into an *optical library* for Modelica.

4.1 Interface

Object oriented modeling requires the encapsulating of the internal function. Communication with other model elements is only allowed through well defined interfaces which are called *connectors* in Modelica [2].

According to the requirements of scalar diffractive optics a connector has to represent a two-dimensional *plane in 3D-space*. Every plane is associated with a complex amplitude light distribution. The spatial discretization of such a cut plane leads to *matrix input/output connectors*; a possible interface structure is shown in Fig. 5.

The propagation of light is not bounded to material and this decoupling suggests the use of *causal* input/output connectors instead of *acausal* connectors as explained in the following.

```

connector InputPlane
  parameter Integer N = 64;      "plane width pixel N"
  parameter Integer M = 64;      "plane height pixel M"
  input Real r_w [3];            "position of cut plane"
  input Real T_w [3, 3];         "orientation of cut plane"
  input Real r [2, N, M];        "position in cut plane"
  input Real E [2, N, M];        "complex light amplitude"
end InputPlane;

connector OutputPlane
  parameter Integer N = 64;      "plane width pixel N"
  parameter Integer M = 64;      "plane height pixel M"
  output Real r_w [3];           "position of cut plane"
  output Real T_w [3, 3];        "orientation of cut plane"
  output Real r [2, N, M];       "position in cut plane"
  output Real E [2, N, M];       "complex light amplitude"
end OutputPlane;
    
```

Fig. 5 Example of an optical input/output connector in Modelica

Modelicas acausal connector principle represents the physical behaviour of components at the cut points. As a consequence the direction of a connection is not specified. This approach follows the interaction between components in the physical world [2].

Generally light does not need material to propagate through space. Light travels through media as well as through vacuum. The lack of an energy carrier for that spatial domain makes it difficult for scalar wave optics to follow the concept of acausal connectors.

Considering the propagation of light in the direction from one component to the next component an optical connection can suitably be described by Modelica signal based or causal connectors as described above.

This component model however does not have the ability to describe a reactive behaviour of light waves to the previous optomechatronic component. This limitation is acceptable for the optical functionality because light waves at one point in space can interpenetrate without disturbing each other.

The influence of light to mechanical components in terms of forces, position and orientation can be neglected for macroscopic mechatronic systems and small optical energy. Under this assumption a light wave can be considered being non-reactive to the mechanical components.

In principle the energy of light can be transformed in heat when interacting with mechanical and electrical components. This conversion however takes place within the optomechatronic components and can be considered by an additional thermal model if needed.

4.2 Components

Every element with a significant optical function is considered being a component. The interesting physical properties include finite mass and spring stiffness and a mechanical connection to other (optical) elements forming a multi-body system (MBS), (Fig. 6).

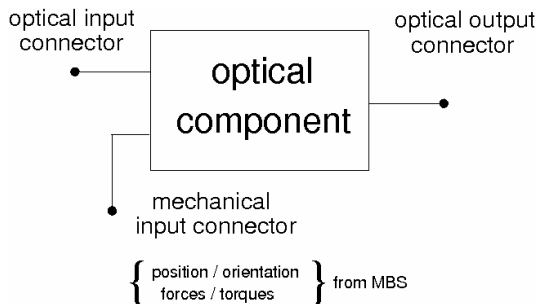


Fig. 6 Generic optical component

For a component model the following approach is chosen. The optical input plane is perpendicularly located directly in front of the optical element body. The output plane is determined by the input plane of the next optical element.

The relative geometry between the actual component and the subsequent component determines the geometry of the output plane, (Fig. 7).

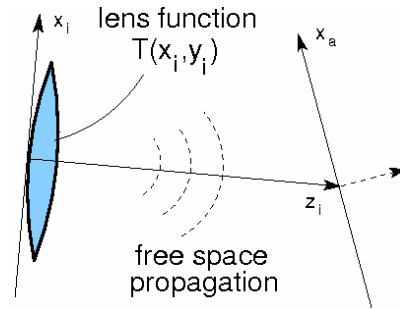


Fig. 7 Optical component functionality

The optical function of the component is defined between input and output plane. It can be modeled as a complex two dimensional optical function $T(x_i, y_i)$ defined at the location of the input plane followed by free space propagation [3], [5].

Matrix calculation is formally supported by Modelica and for many diffractive optical setups the optical functionality can be specified by a complex matrix in a first step.

4.3 Additional properties for an optical library

The implementation of an easy-to-use optical library and the equation based modeling paradigm of Modelica require the description of two dimensional spatial variables in any kind. There is ongoing development for the support of partial differential equations (PDE) and its associated domain definitions which however is not yet operational so far [14], [15]. Hence an equation based description of the optical function $T(x_i, y_i)$ is not possible.

It is therefore necessary to use a physical description of the two-dimensional optical cut plane (optical domain information) and convert it into an internal two-dimensional complex matrix representation which can be handled by the simulation tool, (Fig. 8). The same holds for the description of optical equations which are defined on that domain.

At last the visualisation of two-dimensional data for examination should be available by the simulation tool. Access and visualisation should be granted not only to Modelica connector matrices but also to static internal matrix variables if they exist.

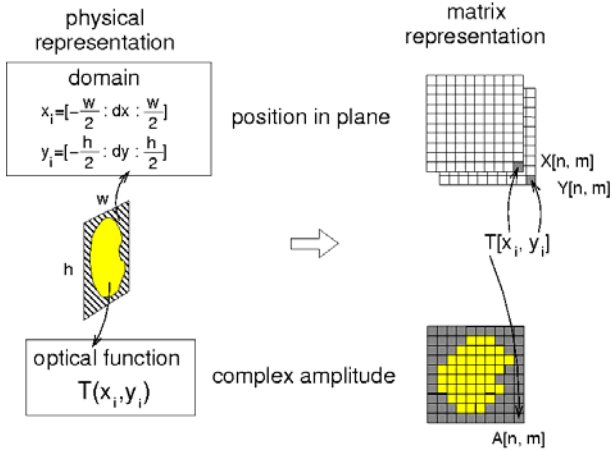


Fig. 8 Domain to matrix conversion

5 Implementation

5.1 Optical Benchmark system: using matrix functionality provided by Modelica

Before designing and implementing complex Modelica library elements, the feasibility and adequate functioning of possible matrix interfaces and internal algorithms must be evaluated. Therefore we consider first the following model of an elementary optical system consisting of a perfectly planar monochromatic light source, a pin-hole aperture and a Fourier lens (Fig. 9). The overall internal component calculations (discrete approximation of angular spectrum method for RAYLEIGH-SOMMERFELD diffraction integral) are simply two complex matrix multiplications (pinhole and lens function) and free space light propagation after the lens.

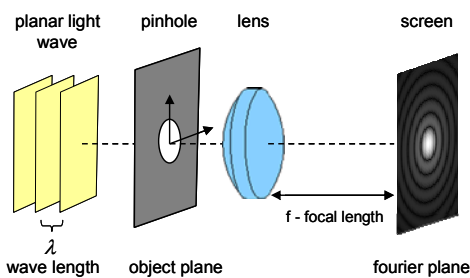


Fig. 9 Optical benchmark system

According to the object oriented modeling approach, the body elements from the existing Modelica multi-body library were extended by optical input and output connectors.

The experiment results for the model translation time (Modelica to C-code) of the benchmark experiments with varying connector matrix size N is shown in Table 1. Increasing matrix size N leads to the observation that the computation time for analyzing and

flattening the model increases significantly. For technically interesting optical resolutions, e.g. $N=1024$, the computation time is out of scope of practical simulation experiments. Using different Modelica based tools like SimulationX[®], OpenModelica or MathModelica[®] show similar behavior.

Table 1. Model translation time (Modelica to C-code) for the benchmark model with SimulationX, matrix size $[N*N]$ ⁺⁾

N	8	16	32	48	64	128
t/ sec	30	40	100	310	1060	-

⁺⁾ Opteron 175, DualCore, 2GB Ram

This behavior is caused by the standard analysis and translation process from MODELICA source code to executable C-code. During the conversion of the model into executable code (flattening process) the tools normally decide which variables become a state variable [2]. This process can take a long analysis time for large matrices.

5.2 Matrix implementation using the External Object / Function interface

A possible alternative *tool independent* solution for the problem described above can be considered using the external function / object interface described in the Modelica language specification [14]. The solution benefits from a fast calculation time, handling of large matrices and can be implemented in a modular way, (Fig. 10).

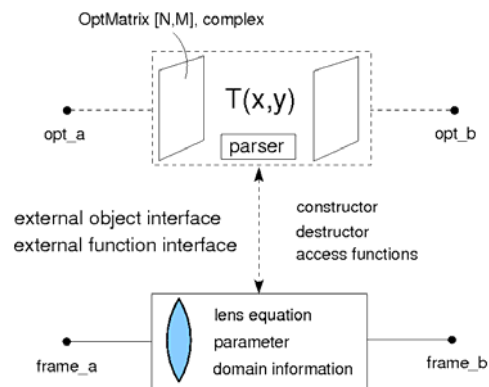


Fig. 10 Basic concept for optical library elements

While handling the matrix calculation externally with C-code the formal description of two-dimensional domains (planes in space), equations and parameter remain in Modelica.

This approach leads to a *composite optical connector* composed of a matrix with complex amplitude values representing the light wave (OptMatrix) and a coordinate frame (input: frame_a, output: frame_b) associated with the respective cut plane, (Fig. 11).

The model OptMatrix is extended from Modelica's external object class. In connection with the coordinate frame which represents the position and orientation of the plane in space it forms the desired optical connector.

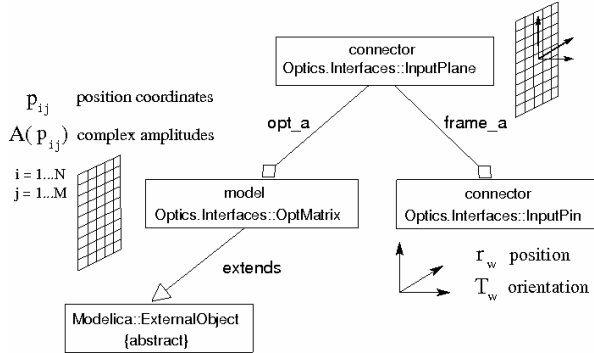


Fig. 11 Optical connector with external object/function interface for the input cut plane

The external object implements the following structure in C-code while constructor and destructor functions handle the memory accordingly:

```
typedef struct {
    unsigned int N; // number of matrix rows
    unsigned int M; // number of matrix cols
    double* AmpReal; // complex amplitude, real part
    double* AmpImag; // complex amplitude, imag. part
    double* PosX; // position in plane, x axis
    double* PosY; // position in plane, y axis
}OptMatrix;
```

The created objects are then handled through the respective Modelica tool. Access can be granted through external functions as described in the Modelica Language Specification [14]. According to the Modelica Language Specification all access functions can be written in C-code, compiled and encapsulated in a system library ('.a' for unix / linux, '.dll' for windows).

Every external function needs a Modelica wrapper function operating with external objects and using the external system library. Those functions are placed in a separate package within the optical Modelica library, (Fig. 12).

Optical components are modeled basically as rigid body elements (maybe within a multibody system) which are extended with optical input and/or output connectors. The generic optical components can be further specified implementing specific optical functionality.

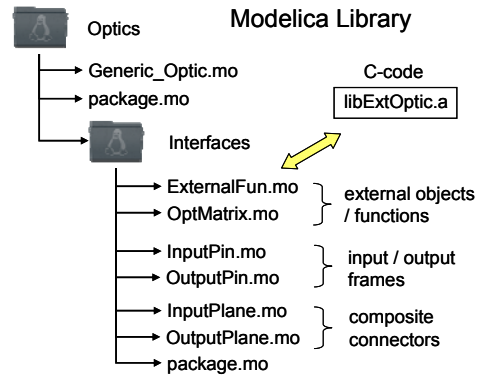


Fig. 12 Modelica optical library structure

Considering again the benchmark model (Fig 9) and its optical functionality it is known that under certain conditions the lens performs the (optical) Fourier-transform of the pinhole circle function [3, 5]. The required conditions are that that pinhole and screen are positioned perpendicular at the focal points of the lens and that the lens is thin and convex.

The Fourier transform of the circle function can be described analytically and it is called Airy function [3]. It will serve as a reference for testing the optical end-to-end performance of the benchmark system and the implementation of the internal diffraction equation.

The first step however will be the proof that handling matrix calculation with Modelica is possible in a manageable way. The test system consists of four optical components, i.e. PlaneWave, PinHole, ThinLens and Screen) leading to the following system structure (Fig 13).

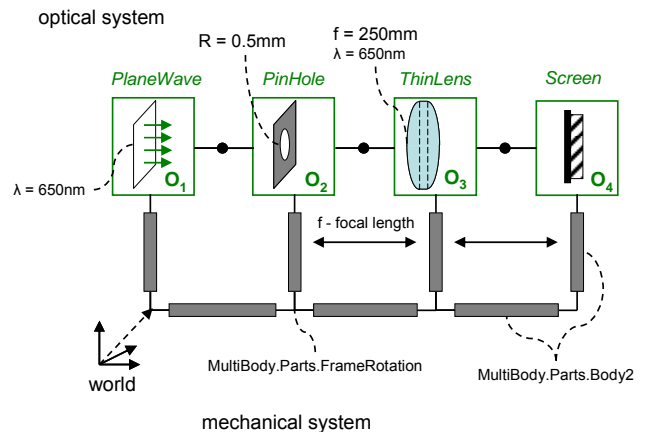


Fig. 13 optical test system in Modelica

For the implementation of the optical functionality we assume perpendicular planes first. This means that the system is static and perfectly aligned with the optical axis as shown in (Fig 13). It allows the use of the angular spectrum method for calculation of the Sommerfeld light diffraction after the pinhole and

the lens according to equation (1). Therefore the following calculation steps are executed internally for those components:

- Matrix multiplication of the respective optical function (pinhole, lens phase function)
- Fast Fourier transformation (FFT) of the input data
- Centering of FFT data (fftshift function)
- Calculation of spatial propagation filter $H(f_x, f_y)$ according to equation (2) and matrix multiplication with FFT data
- Inverse Fourier transformation (iFFT) of the result

For the implementation of the FFT / iFFT the fftw3 library is included in the external function library [18].

The radius of the pinhole ($R=0.5\text{mm}$) as well as typical lens parameters (focal length $f=250\text{mm}$) and optical parameters ($\lambda=650\text{nm}$) are parameterized.

Up to now the simulation experiments show that the overall translation time for the whole model with several complex matrix manipulations and a pure mechanical multi body mass system of an optical bench is acceptable also for large spatial matrix dimensions (Table 2)

Table 2. Model translation time for Benchmark Model with external function interface, matrix size $[N*N]^+$

N	512	1024	2048
t / sec	45	50	55

^{+) Opteron 175, DualCore, 2GB Ram}

A detailed evaluation of simulation performances under dynamic conditions is currently in progress.

5.3 External data access and visualization

The matrix values of every component e.g. input / output planes should be accessible. An easy solution for large matrix data is to store the matrix values as a binary file. Currently the visualization is done by Matlab which reads the binary file over a ‘.mex’ interface and uses image functions for visualization.

6 Conclusion

The physical object oriented modeling and simulation of optomechatronic systems currently lack of an appropriate easy-to-use modeling tool. Modelica can handle electrical, mechanical and thermal physical

domains and provides well implemented libraries but does not cover diffractive optics.

Analysis of scalar wave optics shows that basically matrix calculation is needed to implement an optical functionality and represent cut planes in 3D space. As the basic matrix calculation capability for large matrices is not sufficient by existing Modelica simulation tools the implementation of an optical library needs to incorporate the external object/function interface.

Up to now a concept for optical connectors and components is introduced and a basic optical functionality using FFT algorithms is already implemented and partly tested with Openmodelica and SimulationX [16]. The approach is promising and further implementations of components for an optical library as well as detailed performance and accuracy investigations also for the calculation of diffraction between tilted planes will be executed.

References

- [1] Dyblenko, S. (2009). *Optische Analyse von Bahnwaren mittels , Spekt-ralmethoden - Lösungen und Anwendungen*. Technical report, IPP Symposium, Fakultät Elektrotechnik und Informationstechnik, TU Dresden
- [2] Fritzson, P. (2004). *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*. IEEE Press
- [3] Goodman, J.W. (2005). *Introduction to Fourier Optics*. The McGraw-Hall Companies, 3d edition
- [4] Hardy, J.W. (1998). *Adaptive Optics for Astronomical Telescopes*, Oxford University Press
- [5] Hecht, E. (2002). *Optik*, Oldenburg Wissenschaftsverlag, ISBN 3-486-27359-0
- [6] Janschek, K. (2010). *Systementwurf mechatronischer Systeme: Methoden - Modelle – Konzepte*, Springer.
- [7] Janschek, K., S. Dyblenko, V. Tchernykh, and T. Kaden (2007). *Robuste Verfahren zur Bildaufnahme und Bildauswertung bei Online Messung der Papierformation auf Traversierrahmen*. VDI-Berichte, 1981:57–66.

- [8] Janschek, K., V. Tchernykh and S. Dyblenko (2007). *Performance analysis of optomechatronic image stabilization for a compact space camera*. Control Engineering Practice 15(3 SPEC. ISS.): 333-347
- [9] Janschek, K. and V. Tchernykh (2002). *Optical correlator for image motion compensation in the focal plane of a satellite camera*. Space Technology, 21(4):127-132
- [10] Juday, D.R. and Florence, M.J. (1991). *Full complex modulation with two one-parameter SLMs*, SPIE, vol. 1558, pp. 499-503.
- [11] Sommerfeld A. (1999), *Vorlesungen über theoretische Physik – 4, Optik*. Akad. Verlag Geest u. Portig, 3rd edition
- [12] Tommasi, T and B.Bianco (1992). *Frequency analysis of light diffraction between rotated planes*. Optics Letters, vol. 17, nr. 8
- [13] Matsushima K. (2008). *Formulation of the rotational transformation of wave fields and their application to digital holography*. Applied Optics, vol. 47, nr. 19
- [14] Modelica (2010). The MODELICA Language Specification, Version 3.2, www.modelica.org
- [15] Saldami, L., Bachmann, P. Fritzson, P. and Wiesmann H. (2005). *A Framework for Describing and Solving PDE Models in Modelica*, 4th international Modelica Conference, Hamburg, March 7-8
- [16] Uhlig, A, Beutlich, Blochwitz, Kurzbach and Naehring (2009). *Modellierung und Simulation mit Modelica in SimulationX*, www.iti.de
- [17] Zemax (2010), software for optical system design. www.zemax.com
- [18] <http://www.fft.w.org/>

Transferring Causality Analysis from Synchronous Programs to Hybrid Programs

Kerstin Bauer and Klaus Schneider
 Department of Computer Science
 University of Kaiserslautern
 {k_bauer,klaus.schneider}@cs.uni-kl.de

Abstract

Outputs of synchronous programs may suffer from cyclic dependencies since statements are allowed to read the current outputs' values to determine the actions that generate the current values of the outputs. For this reason, compilers have to perform a causality analysis that ensures that at any point of time, there is a unique and constructive way to determine the outputs. The discrete parts of hybrid systems may suffer from the same problem as observed in synchronous programs. As we recently extended our synchronous language Quartz to describe hybrid systems, we explain in this paper how the causality analysis as originally introduced for synchronous systems can also be used to handle cyclic dependencies in hybrid Modelica programs.

1 Introduction

Reactive systems [20] are systems that have an ongoing interaction with their environment in terms of a discrete sequence of reaction steps. In each reaction step, all the current inputs are read to compute the outputs for the current point in time as well as the system's state for the next point of time.

Synchronous languages [6, 18, 7] such as Esterel [8, 10], Lustre [19], and Quartz [31] have been developed to describe reactive systems. The operational semantics of these languages is defined by so-called *micro and macro steps*, where a macro step consists of finitely many micro steps whose maximal number is known at compile time. Macro steps correspond to reaction steps of the reactive system, and micro steps correspond to atomic actions like assignments of the program. Variables of a synchronous program are *synchronously updated* between macro steps, so that the execution of the micro steps of one macro step is done in the same variable environment of their macro step.

The distinction between micro and macro steps does not only lead to a convenient programming model for reactive systems that allows to efficiently *synthesize hardware and software* as well as a simplified estimation of worst-case reaction times. It is also the key to a *compositional formal semantics* which is a necessary requirement for formal verification and provably correct synthesis procedures. Typically, the semantics are described by means of SOS (structural operational semantics [27]) transition rules that recursively follow the syntax of the program [9, 31].

As synchronous programs are allowed to read their own outputs, mutual dependencies between trigger conditions and the effect of their actions can occur which is also well-known in hardware design [35, 24, 34, 29, 30]. The causality problem is the problem to decide whether such cyclic dependencies can be resolved at runtime for all reachable states and all possible inputs. It is moreover required that for all inputs and all reachable states, there must be a schedule to fire the enabled actions in a sequential schedule so that all values that were required are available when the actions need them. The mere existence of a unique solution of the cyclic equation

systems is thereby not sufficient, the solution must be determined in a constructive way that can be found at runtime. One therefore often speaks of constructive programs [9] that are based on Brouwer's intuitionistic/constructive logic.

In contrast to the discrete reaction steps of an embedded reactive system, its *environment often consists of continuous behaviors* that are determined by the laws of physics. For this reason, the verification of properties that depend on the interaction with the continuous environment requires the consideration of *hybrid systems* (see e.g. [2, 1]). Since most verification problems are undecidable for hybrid systems [22, 21], a *rich theory* of algorithmic, approximative approaches has been developed over the years [28, 25, 23, 16, 17, 3, 13]. However, *only a few languages and tools* that deal with non-trivial hybrid systems [14] are available. Moreover, the languages of most of these tools focus purely on the continuous part of the system providing only little support for modeling and analyzing of the discrete parts of the system [12].

The Modelica language [15, 26] is a highly developed language for the modeling and simulation of hybrid systems. Academic and industrial tools for handling Modelica descriptions together with a vast amount of libraries are available, e.g., the commercial tool Dymola¹ and the academical/industrial tool OpenModelica². However, as it is the case for the other tools, the emphasis of these tools lies on the modeling of the continuous part of the system. Causality cycles in the discrete part of Modelica programs, called algebraic loops, can currently not be handled by these tools.

In this paper, we therefore propose that causality cycles (i.e. algebraic loops) in Modelica programs can be resolved in the same way as done for synchronous programs: By means of a three- or four-valued logic one may perform a causality analysis at compile time that can assure that during runtime, all the cycles can be resolved [32, 33]. To this end, we report about the experiences we made by extending our synchronous programming language Quartz [31] to describe hybrid systems [4, 5]. We have implemented a simulator for the hybrid Quartz language as well as a translation to Modelica programs. We thereby observed that our compiler was able to deal well with causality cycles, while these programs were rejected by tools that are based on Modelica. We see no problem by extending these tools so that they can also benefit from the solutions that are already successfully used for synchronous languages.

2 The Synchronous Language Quartz

Quartz [31] is a synchronous language derived from the Esterel language [11, 8]. The common paradigm of synchronous languages is the perfect synchrony [18, 7] which means that the execution of programs is divided into macro steps that may be interpreted as logical time. As this logical time is the same in all concurrent threads, these threads run in lockstep, which leads to a deterministic form of concurrency. Macro steps are divided into finitely many micro steps that are atomic actions of the programs. Moreover, variables change synchronously in macro steps, i.e., variables have unique values in each macro step.

In the following, we only give a brief overview of Quartz, and refer to [31] for further details. Provided that S , S_1 , and S_2 are statements, ℓ is a location variable, x is a variable, σ is a Boolean expression, and ρ is a type, then the following are statements (parts given in square brackets are optional):

- nothing (empty statement)
- $x = \tau$ and $\text{next}(x) = \tau$ (assignments)

¹<http://www.dymola.com>

²<http://www.openModelica.org>

- `assume(φ)` and `assert(φ)` (assumptions and assertions)
- `ℓ : pause` (start/end of macro step)
- `if (σ) S_1 else S_2` (conditional)
- `$S_1; S_2$` (sequences)
- `do S while(σ)` (loops)
- `$S_1 \parallel S_2$` (synchronous concurrency)
- `[weak] [immediate] abort S when(σ)`
- `[weak] [immediate] suspend S when(σ)`
- `{ ρ x ; S }` (local variable x of type ρ)

The pause statement defines a control flow location ℓ – a boolean variable being true iff the control flow is currently at `ℓ : pause`. Since all other statements are executed in zero time, the control flow only rests at these positions in the program, and thus the possible control flow states are the subsets of the set of locations.

There are two variants of assignments that both evaluate the right-hand side τ in the current macro step (variable environment). While immediate assignments `$x = \tau$` immediately transfer the value of τ to the left-hand side x , delayed assignments `$\text{next}(x) = \tau$` transfer this value only in the following step.

If the value of a variable is not determined by assignments, a default value is computed according to the declaration of the variable. To this end, declarations consist of a *storage class in addition to the type* of a variable. There are two storage classes, namely `mem` and `event` that choose the previous value (mem variables) or a default value (event variables) in case no assignment determines the value of a variable. Available data types are booleans, bitvectors, signed/unsigned bounded/unbounded integers, arrays and tuples.

In addition to the statements known from other imperative languages (conditionals, sequences and loops), Quartz offers synchronous concurrency `$S_1 \parallel S_2$` and sophisticated preemption and suspension statements, as well as many more statements like generic statements to allow comfortable descriptions of reactive systems (see [31] for the complete syntax and semantics).

Our Averest system³ provides algorithms that translate a synchronous program to a set of guarded actions [31], i.e., pairs (γ, α) consisting of a trigger condition γ and an action α . Actions are thereby assignments `$x = \tau$` and `$\text{next}(x) = \tau$` , assumptions `$\text{assume}(\varphi)$` , or assertions `$\text{assert}(\varphi)$` . The meaning of a guarded action is obvious: in every macro step, all actions are executed whose guards are true. Thus, it is straightforward to construct a symbolic representation of the transition relation in terms of the guarded actions (see [31]).

3 Causality Analysis in System Modeling of Controllers

3.1 Causality Problems in Quartz

As already mentioned, synchronous programs often suffer from cyclic dependencies since the programs are allowed to read their own outputs for determining these outputs. It is simple to determine whether a program has such cyclic dependencies by means of a static syntactic analysis. If no cycles occur, it is straightforward to generate code that exactly implements the given program's behavior. However, if cycles occur, the programs can get stuck in deadlocks or may implement one of many possible nondeterministic behaviors. For this reason, a causality analysis has to be performed that checks whether the cycles can be resolved during runtime for all possible states and inputs.

To explain the causality analysis performed in compilers for synchronous languages, consider the execution of a Quartz program. To this end, we assume that

³<http://www.averest.org>

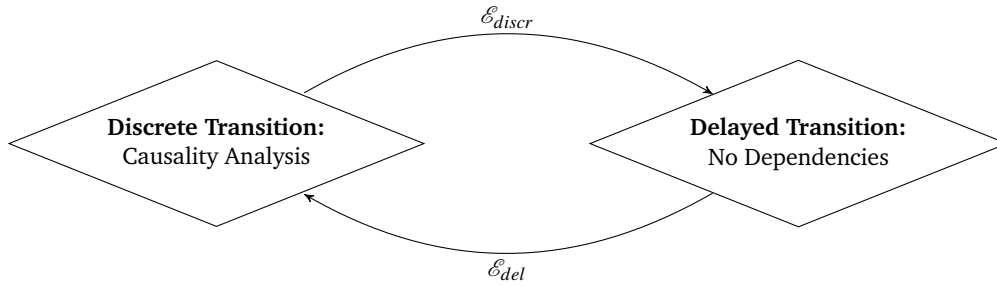


Figure 1: Execution of a Macro Step of Quartz

program	guarded actions
<pre> module P(event ?i,o1,o2) { if(i) { if(o1) o2=true; } else { if(o2) o1=true; } assert(!o1 & !o2); } </pre>	<pre> regular guarded actions: i & o1 ==> o2=true; !i & o2 ==> o2=true; true ==> assert(!o1 & !o2); reaction to absence actions: !(i & o1) ==> o2=false; !(i & o2) ==> o2=false; </pre>

Figure 2: Example Quartz Program with a Causality Cycle and its Translation to Guarded Actions.

the program has already been compiled in a set of guarded actions (γ, α) as explained in the previous section. The execution of such a set of guarded actions is then best explained by a discrete two-location automaton as shown in Figure 1. Each macro step starts with a partial environment \mathcal{E}_{del} that is determined by the delayed actions of the previous macro step⁴. After assigning this partial environment to the current discrete environment \mathcal{E}_{discr} , it is checked which trigger conditions become true so that further actions can be executed. Also, actions whose trigger condition become false, can be singled out. Due to the execution of new actions, more values become known, and the same procedure is repeated until no more values become known. The program is constructive (i.e. causally correct) if the environment \mathcal{E}_{discr} is fully defined at the end. Finally, the delayed actions are executed to provide a new delayed environment \mathcal{E}_{del} for the next execution step. As all values are known at that point of time, the computation of \mathcal{E}_{del} does not require a causality analysis.

A simulator can directly implement the procedure outlined above. A compiler has to perform this analysis for all inputs, which is typically done in a symbolic way (similar to model-checking) to avoid the enumeration of all states and inputs. Although it is possible to remove the causality cycles if the programs are constructive, it is often better to retain them in the code, since it is known that the cyclic code will be typically smaller [24, 29, 30] and will not have problems during runtime (if the code is executed as outlined above).

A simple example for a Quartz program with cyclic dependencies is shown in Figure 2. The program P has a boolean input *i* and two boolean outputs *o1* and *o2* that can also be read. As *o1* and *o2* are boolean event variables, they are reset to their default value `false` whenever there is no action setting them actively.

The translation to guarded actions yields the guarded actions shown on the right of Figure 2. We distinguish between the ‘regular’ guarded actions that are those that appear in the program, and additional guarded actions that are added by the compiler to reset event variables or to store memorized variables.

⁴In the first macro step initial assignments are given instead.

Iteration Step	i	o1	o2	May-Set of Actions	Must-Set of Actions
1	true	\perp	\perp	$o1 \Rightarrow o2 = \text{true}$ $\neg o1 \Rightarrow o2 = \text{false}$ $\neg o2 \Rightarrow o1 = \text{false}$	$\text{true} \Rightarrow o1 = \text{false}$
2	true	false	\perp		$\text{true} \Rightarrow o2 = \text{false}$ $\text{true} \Rightarrow o1 = \text{false}$
3	true	false	false		$\text{true} \Rightarrow o2 = \text{false}$ $\text{true} \Rightarrow o1 = \text{false}$

 Table 1: Fixpoint Iteration for Input Value $i = \text{true}$

The causality analysis for input $i=\text{true}$ is shown in Table 1. It can be seen that the values of all variables can be determined within three steps even though $o1$ and $o2$ depend on each other. Depending on the so-far obtained partial variable environment, the set of guarded actions is partitioned into ‘must actions’ that must be executed (their trigger condition is true), ‘cannot actions’ that cannot be executed (their trigger condition is false), and the remaining ‘may actions’ whose might or might not be executed (their trigger condition is neither true nor false due to still unknown variables). The value \perp shown in Table indicates that currently no valid value is known for the corresponding variable.

A similar fixpoint iteration can be shown for $i = \text{false}$. However, in this case, the value of $o2$ will be computed before the one of $o1$, so that a different schedule has to be used depending on input i .

```

model causality
  Real t      (start = 0);
  Boolean i   (start = false);
  Boolean o1  (start = false);
  Boolean o2  (start = false);
equation
  der(t) = 1.0;
  when { t>=0.1 } then
    i = if pre(i) then false else true;
    o1 = if (not(i) and o2) then true else false;
    o2 = if (i and o1) then true else false;
    reinit(t,0.0);
  end when;
end causality;
    
```

Figure 3: Modelica Program with Causality Conflict

3.2 Causality Problems in Modelica

The Modelica language also supports the synchronous model of computation in its discrete part. Because of this, the guarded actions obtained from Quartz programs can be easily translated to equivalent Modelica programs. However, the tools we used were not able to deal with causality problems (algebraic loops), illustrated with the Modelica program shown in Figure 3. This program is obtained by translating the guarded actions of the Quartz program P to Modelica. The real variable t is only a time trigger and does not influence the discrete behavior of the program.

Neither OpenModelica 1.5.0 nor the demoverion of the industrial tool Dymola 7 are able to execute the program, because of the occurring algebraic loop. Even when simplifying the line $i = \text{if pre}(i) \text{ then false else true};$ by $i = \text{true};$ such that there exists a unique computation order as shown in Table 1, the tools cannot handle the program. Failure reports of Dymola simply note: "Current version cannot generate code for an algebraic loop involving integers or Boolean".

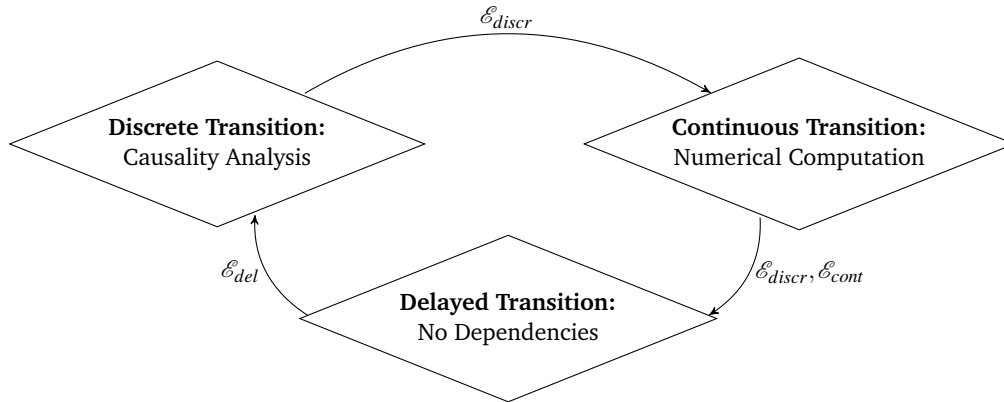


Figure 4: Execution of a Macro Step of Hybrid Quartz

4 Hybrid Quartz

The environments of embedded reactive systems are often defined by continuous behaviors that are determined by the laws of physics. To be able to describe these continuous environments, the synchronous language Quartz has been recently extended to Hybrid Quartz [4]. While time in synchronous languages is purely logical, hybrid systems require the consideration of physical time. In order to combine these inherently different time concepts, the computational model of macro steps is endowed by a continuous transition as depicted in Figure 4.

In addition to the memorized and event variables, there are additionally hybrid variables in Hybrid Quartz. Hybrid variables must have the data type `real` and are the only variables who have continuous evolutions during the macro steps. These continuous variables can be assigned by *flow assignments* $x \leftarrow \tau$ whose left hand side may also refer to the derivation of the continuous variable $\text{drv}(x) \leftarrow \tau$. For specification and verification reasons, one can additionally impose constraints $\text{constrain}\{S, M, E\}(\phi)$, which state that the condition ϕ has to be satisfied at the Start, at any interMediate point or at the End of the continuous evolution.

The continuous actions $x \leftarrow \tau$, $\text{drv}(x) \leftarrow \tau$, and $\text{constrain}\{S, M, E\}(\phi)$ may only occur in special statements of the form `flow S until(σ)` where S is a list of flow assignments and σ is a so-called *release condition* that terminates the continuous phase defined by the flow statement. The compiler also generates guarded actions for Hybrid Quartz, where the actions now additionally consist of continuous guarded actions, i.e., guarded actions that contain flow assignments or constrain actions as well as guarded actions ($\gamma, \text{release}(\sigma)$) for the release conditions.

In Hybrid Quartz, there exists also a new operator `cont` that allows one to access the discrete as well as the continuous value of a hybrid variable during a flow-statement, i.e., its value at the time when the continuous phase was started and its value at some considered point of time during the continuous phase.

More information on hybrid Quartz can be found in [4].

The simulation of Hybrid Quartz programs, depicted in Figure 4, is performed as follows: After computing the discrete variable environment \mathcal{E}_{discr} by means of the causality analysis, one can determine which of the continuous actions are enabled. Taking the values of \mathcal{E}_{discr} as initial values for potential systems of ordinary differential equations, the continuous flow of the macro step is executed until the first active release condition σ is satisfied. \mathcal{E}_{cont} stores the values of all variables at the end of the continuous transition. Note, that \mathcal{E}_{discr} and \mathcal{E}_{cont} coincide in all discrete variables, as these do not change their value during the continuous evolution. The delayed transition now obtains both environments as inputs and computes the partial environment \mathcal{E}_{del} for the following macro step.

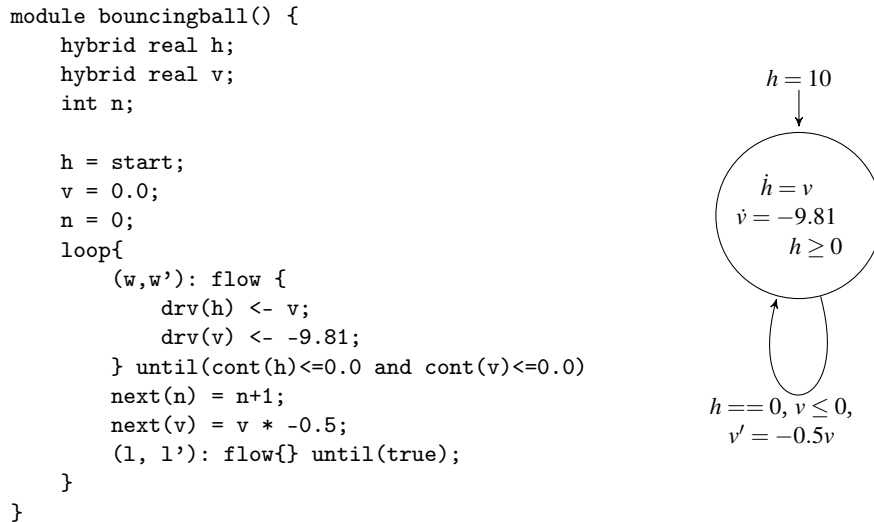


Figure 5: Bouncing Ball Example in Hybrid Quartz

We conclude this section by listing a Quartz program for the well-known Bouncing Ball example in Figure 5. The program models a bouncing ball, whose velocity is reduced by half whenever it bounces on the floor. Initially, the ball starts at height $h = 0.0$ with velocity $v = 0.0$, and no bounces appeared so far ($n = 0$).

5 From Hybrid Quartz to Modelica Programs

The intermediate code format of Hybrid Quartz in the form of guarded actions is already very close to Modelica code. Only few adjustments need to be made in order to obtain an equivalent Modelica program. The translated Modelica program will have the general form

```

model fromQuartz
  Variable Declarations;
equation
  all continuous guarded actions
  when (disjunction of all guarded release conditions)
    for each variable: gather all guarded actions in one expression
    for each continuous variable: reinit with the associated discrete value
  end when;
end fromQuartz:
    
```

Hybrid Quartz essentially provides two environments for each macro step, \mathcal{E}_{discr} and \mathcal{E}_{cont} . The third environment \mathcal{E}_{del} can be omitted in Modelica models, as one can use the pre-operator instead.

To obtain an equivalent Modelica model, it is necessary to represent these two environments in some way. The easiest way is to create for each variable a discrete version and a continuous version, where the discrete version refers to \mathcal{E}_{discr} and the continuous version refers to \mathcal{E}_{cont} . As the two environments coincide on all discrete variables, it suffices to only create a copy for each hybrid variable. That means, each hybrid variable `hybrid real x` in Quartz is replaced in Modelica by

```
discrete Real x_discr; Real x;
```

whereas each discrete⁵ variable type `y` in Quartz is replaced by

```
discrete type y;.
```

⁵The storage class of discrete variables is handled by the compiler in that appropriate reactions to absence are generated.

The guarded actions are translated into Modelica in two successive steps: In the first step, each guarded action $\gamma \Rightarrow \alpha$ is translated into the equivalent statement

$$\text{if } \sigma \text{ then } \alpha.$$

As this code is not yet supported by the existing tools, these guarded actions will be gathered together in actions of the form

$$x = \text{if } \gamma_1 \text{ then } \alpha_1 \text{ else if } \dots$$

which are accepted by both Dymola and OpenModelica.

The first type of actions are immediate assignments in the form of $\gamma \Rightarrow x = \tau$. This action is replaced by

$$\text{if } \gamma_{discr} \text{ then } x = \tau_{discr}$$

where γ_{discr} , τ_{discr} replace each occurrence of a continuous variable x by either x or x_discr , depending on whether x lies within a `cont(_)` statement or not.

The second type of guarded actions is that of delayed actions $\gamma \Rightarrow \text{next}(x) = \tau$. Analogously to the discrete actions, first γ and τ are replaced by γ_{discr} and τ_{discr} . In a second step, all variables x are replaced by `pre(x)` in γ_{discr} as well as in τ_{discr} , as the computation of these actions must be done w.r.t. the variable environment of the previous macro step. This has the same effect as computing the delayed assignments in the current macro step and storing them in an intermediate environment.

Expressions within flow actions conditions are treated analogously.

In the second step, for each variable all conditional assignments are collected and written as a single assignment with the variable on the left hand side. Additionally, according to the storage class of the variable, the reaction to absence is encoded:

- event variables are set to their default value.
- memorized variables are set to their previous discrete value.
- hybrid variables are set to their last known continuous value.

Equations concerning the continuous flow are written first in the equation setting. As condition for the following when-statement, the boolean disjunction of all release actions is given.

According to simple test models, Modelica or at least the demo version of Dymola and OpenModelica are not capable to handle execution steps, where the continuous evolution actually does not consume physical time. Therefore we must add a timer t which is reinitialized to 0 during each discrete transition and increases linearly in time. All activation conditions of the when statement now must be conjuncted with the condition that time has actually advanced, i.e. $t \geq min$, where min is some minimal time (chosen suitable small).

Within the when-statement, now all discrete equations are written down. Furthermore, all hybrid variables are reinitialized with their discrete values.

The translation procedure is finally illustrated by the bouncing ball example given in Figure 5. In a first step, the program is compiled to guarded actions. The boolean variable `Init` will hold iff the execution is in the initial step. Furthermore, in order to increase readability, the boolean expression $\text{cont}(h) \leq 0.0$ and $\text{cont}(v) \leq 0.0$ in the flow statement is replaced by σ . The corresponding Modelica model is given in Figure 6.

1. $(\text{Init} \vee 1) \wedge \sigma \Rightarrow \text{next}(w) = \text{true}$
2. $(w' \vee \text{Init} \vee 1) \wedge \neg \sigma \Rightarrow \text{next}(w') = \text{true}$
3. $w \Rightarrow \text{next}(1) = \text{true}$
4. $\text{Init} \vee 1 \vee w' \Rightarrow \text{der}(h) <- v$
5. $\text{Init} \vee 1 \vee w' \Rightarrow \text{der}(v) <- -9.81$
6. $w \Rightarrow \text{next}(n) = n+1$
7. $w \Rightarrow \text{next}(v) = v*-0.5$
8. $\text{Init} \vee 1 \vee w' \Rightarrow \text{release}(\sigma)$

```

model BouncingBall
  Real h (start = 10);
  discrete Real h_discr (start = 0);
  Real v;
  discrete Real v_discr (start = 0);
  Boolean Init(start = true);
  Boolean w(start = false);
  Boolean w2(start = false);
  Boolean l (start = false);
  Integer n (start = 0);
  Real t(start=0);

equation
  der(t) = 1;
  der(h) = if (Init or l or w2) then v else 0;
  der(v) = if (Init or l or w2) then -9.81 else 0;

  when ((t>=0.01) and (((Init or l or w2) and h < 0 and v < 0) or w)) then
    n = if pre(w) then pre(n) + 1 else pre(n);
    v_discr = if pre(w) then - pre(v) * 0.5 else pre(v);
    h_discr = pre(h);
    w = if (pre(Init) or pre(l) or pre(w2)) and pre(v) <= 0 and pre(h) <= 0
        then true else false;
    w2 = if (pre(Init) or pre(l) or pre(w2)) and not (pre(v) <= 0 and pre(h) <= 0)
        then true else false;
    l = if pre(w) then true else false;
    Init = false;

    reinit(v, v_discr);
    reinit(h, h_discr);
    reinit(t, 0);

  end when;
end BouncingBall;

```

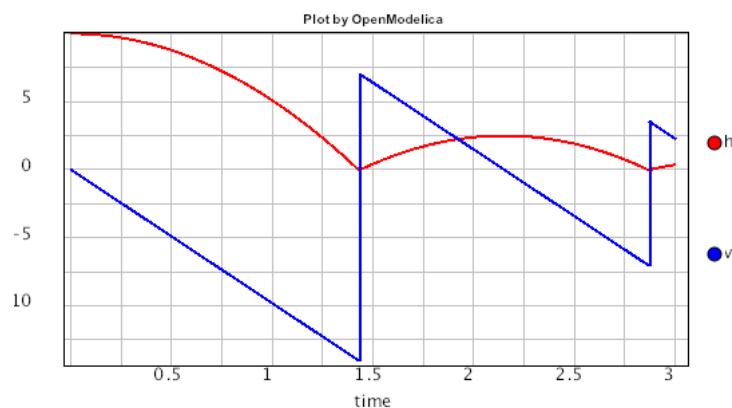


Figure 6: Modelica Model of the Bouncing Ball generated from a Hybrid Quartz Program

6 Conclusions

Hybrid Quartz and Modelica approach the modeling of hybrid systems from different starting points: While Hybrid Quartz emphasizes mainly the discrete part of the system, the Modelica language puts its emphasis on the continuous part. Nevertheless, this paper shows that both languages not only share a common core, but that one can translate Hybrid Quartz programs one-to-one to Modelica programs.

However, due to the different origins of the two modeling languages, Hybrid Quartz and Modelica provide very different algorithms for the analysis and simulation of the hybrid programs: Based on the well-developed theory within the discrete domain, Quartz (like other synchronous languages) is able to solve non-trivial algebraic loops, thus allowing a much broader variety of models. As non-trivial causal dependencies may easily occur, this is a big advantage over Modelica programs. Furthermore, discrete Quartz already provides algorithms for formal verification.

On the other hand, as the main emphasis of Modelica lies on the continuous part of the hybrid system, algorithms to deal with the continuous evolutions of such systems are well-developed. As Quartz only has been recently extended to Hybrid Quartz, sophisticated algorithms for dealing with the continuous dynamics are still missing.

Thus, tools for both languages could learn a lot from each other. Tools for synchronous languages offer sophisticated procedures for formal verification, worst case execution time analysis and causality analysis, while tools that deal with Modelica offer much better support for continuous dynamics. Thus, the resulting mixture could be a lot more powerful as both tools on their own.

References

- [1] R. Alur, C. Courcoubetis, N. Halbwachs, T. Henzinger, P. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science (TCS)*, 138(1):3–34, 1995.
- [2] R. Alur, C. Courcoubetis, T. Henzinger, and P.-H. Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In R. Grossmann, A. Nerode, A. Ravn, and H. Rischel, editors, *Hybrid Systems*, volume 736 of *LNCS*, pages 209–229. Springer, 1993.
- [3] R. Alur, T. Henzinger, G. Lafferriere, and G. Pappas. Discrete abstractions of hybrid systems. *Proceedings of the IEEE*, 88(7):971–984, 2000.
- [4] K. Bauer and K. Schneider. From synchronous programs to symbolic representations of hybrid systems. In K. Johansson and W. Yi, editors, *Hybrid Systems: Computation and Control (HSCC)*, pages 41–50, Stockholm, Sweden, 2010. ACM.
- [5] K. Bauer and K. Schneider. Predicting events for the simulation of hybrid systems. In *International Conference on Embedded Software and Systems (ICES)*, Bradford, United Kingdom, 2010. IEEE Computer Society.
- [6] A. Benveniste and G. Berry. The synchronous approach to reactive real-time systems. *Proceedings of the IEEE*, 79(9):1270–1282, 1991.
- [7] A. Benveniste, P. Caspi, S. Edwards, N. Halbwachs, P. Le Guernic, and R. de Simone. The synchronous languages twelve years later. *Proceedings of the IEEE*, 91(1):64–83, 2003.
- [8] G. Berry. The foundations of Esterel. In G. Plotkin, C. Stirling, and M. Tofte, editors, *Proof, Language and Interaction: Essays in Honour of Robin Milner*. MIT Press, 1998.
- [9] G. Berry. The constructive semantics of pure Esterel. <http://www-sop.inria.fr/esterel.org/>, July 1999.
- [10] G. Berry. The Esterel v5 language primer. <http://www-sop.inria.fr/esterel.org/>, July 2000.
- [11] G. Berry and G. Gonthier. The Esterel synchronous programming language: Design, semantics, implementation. *Science of Computer Programming*, 19(2):87–152, 1992.
- [12] X. Briand and B. Jeannot. Combining control and data abstraction in the verification of hybrid systems. In R. Bloem and P. Schaumont, editors, *Formal Methods and Models for Codesign (MEMOCODE)*, pages 141–150, Cambridge, Massachusetts, USA, 2009. IEEE Computer Society.

- [13] D. Campagna and C. Piazza. Hybrid automata in systems biology: How far can we go? *Electronic Notes in Theoretical Computer Science (ENTCS)*, 229:93–108, 2009.
- [14] L. Carloni, M. Di Benedetto, R. Passerone, A. Pinto, and A. Sangiovanni-Vincentelli. Modeling techniques, programming languages, and design toolsets for hybrid systems, 2004. Report on the Columbus Project, <http://www.columbus.gr>.
- [15] P. Fritzson and V. Engelson. Modelica - a unified object-oriented language for system modeling and simulation. In *Object-Oriented Programming*, volume 1445 of *LNCS*, pages 67–90. Springer, 1998.
- [16] M. Fränzle. What will be eventually true of polynomial hybrid automata? In N. Kobayashi and B. Pierce, editors, *Theoretical Aspects of Computer Software (TACS)*, volume 2215 of *LNCS*, pages 340–359, Sendai, Japan, 2001. Springer.
- [17] R. Ghosh, A. Tiwari, and C. Tomlin. Automated symbolic reachability analysis with application to delta-notch signaling automata. In O. Maler and A. Pnueli, editors, *Hybrid Systems: Computation and Control (HSCC)*, volume 2623 of *LNCS*, pages 233–248, Prague, Czech Republic, 2003. Springer.
- [18] N. Halbwachs. *Synchronous programming of reactive systems*. Kluwer, 1993.
- [19] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The synchronous dataflow programming language LUSTRE. *Proceedings of the IEEE*, 79(9):1305–1320, September 1991.
- [20] D. Harel. Statecharts: A visual formulation for complex systems. *Science of Computer Programming*, 8(3):231–274, 1987.
- [21] T. Henzinger. The theory of hybrid automata. In *Logic in Computer Science (LICS)*, pages 278–292, New Brunswick, New Jersey, USA, 1996. IEEE Computer Society.
- [22] T. Henzinger, P. Kopke, A. Puri, and P. Varaiya. What’s decidable about hybrid automata? In *Las Vegas, Nevada, USA*, pages 373–382. ACM, 1995.
- [23] G. Lafferriere, G. Pappas, and S. Yovine. A new class of decidable hybrid systems. In F. Vaandrager and J. van Schuppen, editors, *Hybrid Systems: Computation and Control (HSCC)*, volume 1569 of *LNCS*, pages 137–151, Berg en Dal, The Netherlands, 1999. Springer.
- [24] S. Malik. Analysis of cycle combinational circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (T-CAD)*, 13(7):950–956, July 1994.
- [25] J. Miller. Decidability and complexity results for timed automata and semi-linear hybrid automata. In N. Lynch and B. Krogh, editors, *Hybrid Systems: Computation and Control (HSCC)*, volume 1790 of *LNCS*, pages 296–309, Pittsburgh, Pennsylvania, USA, 2000. Springer.
- [26] Modelica Association. Modelica - a unified object-oriented language for physical systems modeling, language specification version 2.0, 2002. <http://www.Modelica.org>.
- [27] G. Plotkin. A structural approach to operational semantics. Technical Report FN-19, DAIMI, Aarhus, Denmark, 1981.
- [28] S. Ratschan and Z. She. Safety verification of hybrid systems by constraint propagation based abstraction refinement. In M. Morari and L. Thiele, editors, *Hybrid Systems: Computation and Control (HSCC)*, volume 3414 of *LNCS*, pages 573–589, Zürich, Switzerland, 2005. Springer.
- [29] M. Riedel and J. Bruck. Cyclic combinational circuits: Analysis for synthesis. In *International Workshop on Logic and Synthesis (IWLS)*, Laguna Beach, California, USA, 2003.
- [30] M. Riedel and J. Bruck. The synthesis of cyclic combinational circuits. In *Design Automation Conference (DAC)*, pages 163–168, Anaheim, California, USA, 2003. ACM.
- [31] K. Schneider. The synchronous programming language Quartz. Internal Report 375, Department of Computer Science, University of Kaiserslautern, Kaiserslautern, Germany, 2009.
- [32] K. Schneider and J. Brandt. Performing causality analysis by bounded model checking. In *Application of Concurrency to System Design (ACSD)*, pages 78–87, Xi’an, China, 2008. IEEE Computer Society.
- [33] K. Schneider, J. Brandt, T. Schuele, and T. Tuerk. Maximal causality analysis. In J. Desel and Y. Watanabe, editors, *Application of Concurrency to System Design (ACSD)*, pages 106–115, St. Malo, France, 2005. IEEE Computer Society.
- [34] T. Shiple, G. Berry, and H. Touati. Constructive analysis of cyclic circuits. In *European Design and Test Conference (EDTC)*, Paris, France, 1996. IEEE Computer Society.
- [35] L. Stok. False loops through resource sharing. In *International Conference on Computer-Aided Design (ICCAD)*, pages 345–348. IEEE Computer Society, 1992.

Integration of CasADi and JModelica.org

Joel Andersson^c Johan Åkesson^{a,b} Francesco Casella^d Moritz Diehl^c

^aDepartment of Automatic Control, Lund University, Sweden

^bModelon AB, Sweden

^cDepartment of Electrical Engineering and Optimization in Engineering Center (OPTEC),
K.U. Leuven, Belgium

^dDipartimento di Elettronica e Informazione, Politecnico di Milano, Italy

Abstract

This paper presents the integration of two open source softwares: CasADi, which is a framework for efficient evaluation of expressions and their derivatives, and the Modelica-based platform JModelica.org. The integration of the tools is based on an XML format for exchange of DAE models. The JModelica.org platform supports export of models in this XML format, whereas CasADi supports import of models expressed in this format. Furthermore, we have carried out comparisons with ACADO, which is a multiple shooting package for solving optimal control problems.

CasADi, in turn, has been interfaced with ACADO Toolkit, enabling users to define optimal control problems using Modelica and Optimica specifications, and use solve using direct multiple shooting. In addition, a collocation algorithm targeted at solving large-scale DAE constrained dynamic optimization problems has been implemented. This implementation explores CasADi's Python and IPOPT interfaces, which offer a convenient, yet highly efficient environment for development of optimization algorithms. The algorithms are evaluated using industrially relevant benchmark problems.

Keywords: Dynamic optimization, Symbolic manipulation, Modelica, JModelica.org, ACADO Toolkit, CasADi

1 Introduction

High-level modeling frameworks such as Modelica are becoming increasingly used in industrial applications. Existing modeling languages enable users to rapidly develop complex large-scale models. Traditionally, the main target for such models has been simulation, i.e., to define virtual experiments where a simulation software computes the model response.

During the last two decades, methods for large scale dynamic optimization problems have been developed. Notably, linear and non-linear model predictive control (MPC) have had a significant impact in the industrial community, in particular in the area of process control. In MPC, an optimal control problem is solved for a finite horizon, and the first optimal control interval is applied to the plant. At the next sample, the procedure is repeated, and the optimal control problem is solved again, based on updated state estimates. The advantages of MPC as compared to traditional control strategies are that it takes into account state and input constraints and that it handles systems with multiple inputs and multiple outputs. Also, MPC offers means to trade performance and robustness by tuning of a cost function, where the importance of different, often contradictory, control objectives are encoded. A bottleneck when applying MPC strategies, in particular in the case of non-linear systems, is that the computational effort needed to solve the optimal control problem in each sample is significant. Development of algorithms compatible with the real-time requirements of MPC has therefore been a strong theme in the research community, see e.g., [15, 40].

Driven by the impact of high-level modeling languages in the industrial community, there have been several efforts to integrate frameworks for such languages with algorithms for dynamic optimization. Examples include gPROMS [34], which supports dynamic optimization of process systems models and Dymola [13], which supports parameter and design optimization of Modelica models. Several other applications of dynamic optimization of Modelica models have been reported, e.g., [20, 35, 3, 33, 28].

This paper reports results of an effort where three different open source packages have been integrated: JModelica.org, [2], ACADO Toolkit, [26], and CasADi, [4]. The integration relies on the XML

model exchange format previously reported in [32]. Two main results are presented in the paper. Firstly, it is shown how CasADi, supporting import of the mentioned XML model format, has been used to integrate JModelica.org with ACADO Toolkit. Secondly, a novel direct collocation method has been developed based on the innovative symbolic manipulation features of CasADi. From a user's perspective, both CasADi and JModelica.org come with Python interfaces, which makes scripting, plotting and analysis of results straightforward.

The benefit of integrating additional algorithms for solution of dynamic optimization problems in the JModelica.org platform is that users may experiment with different algorithms and choose the one that is most suited for their particular problem. To some extent, the situation is similar to that of choosing an integrator for a simulation experiment, where it is well known that stiff systems require more sophisticated solvers than non-stiff systems.

The paper is organized as follows: in Section 2, background on dynamic optimization, Modelica and Optimica, ACADO and JModelica.org is given. Section 3 describes CasADi and recent extensions thereof. Section 4 reports a novel Python-based collocation implementation and in Section 5, benchmark results are presented. The paper ends with a summary and conclusions in Section 6.

2 Background

2.1 Dynamic optimization

Dynamic optimization is the solution of decision making problems constrained by differential or differential-algebraic equations. A common formulation is the optimal control problem (OCP) based on differential-algebraic equations (DAE) on the form

$$\begin{aligned} \min_{x,u,z,p} \int_0^T l(t,x(t),\dot{x}(t),z(t),u(t),p) dt \\ + E(T,x(T),z(T),p) \\ \text{subject to} \\ f(t,x(t),\dot{x}(t),z(t),u(t),p) = 0 \quad t \in [0,T] \\ h(t,x(t),\dot{x}(t),z(t),u(t),p) \leq 0 \quad t \in [0,T] \\ x(0) = x_0 \\ u_{\min} \leq u(t) \leq u_{\max} \quad t \in [0,T] \\ p_{\min} \leq p \leq p_{\max} \end{aligned} \quad (1)$$

where $x \in \mathbf{R}^{N_x}$ and $z \in \mathbf{R}^{N_z}$ denote differential and algebraic states respectively, $u \in \mathbf{R}^{N_u}$ are the free con-

trol signals and $p \in \mathbf{R}^{N_p}$ a set of free parameters in the model. The DAE is represented by the $N_x + N_z$ equations $f(t,x(t),\dot{x},z(t),u(t),p) = 0$, with the initial value for x explicitly given.

The objective function consists of an integral cost contribution (or Lagrange term) and an end time cost contribution (or Mayer term). The time horizon $[0, T]$ may or may not be fixed.

Numerical methods for solving this optimization problem emerged with the birth of the electronic computer in the 1950's and were typically based on either *dynamic programming*, which is limited to very small problems, or methods based on the calculus of variation, so-called *indirect methods*. The inability of indirect methods to deal with inequality constraints, represented above as the path constraint $h(t,x(t),\dot{x},z(t),u(t),p) \leq 0$ and the control bounds $u(\cdot) \in [u_{\min}, u_{\max}]$, shifted the focus in the early 1980's to *direct-methods*, where instead the control, and (possibly) the state, trajectories are parametrized to form a finite-dimensional non-linear program (NLP), for which standard solution methods exist. In this work, we employ two of the most popular methods in this field, namely *direct multiple shooting* and *direct collocation*, see [8, 9] for an overview.

2.1.1 Direct multiple shooting

After parametrizing the control trajectories, for example by using a piecewise constant approximation, the time varying state trajectories can be eliminated by making use of standard ODE or DAE integrators. This method of embedding DAE integrators in the NLP formulation is referred to as *single shooting*. The advantage is that it makes use of two standard problem formulations, the solution of initial value problems for DAEs and the solution of unstructured NLPs. For both of these problems, there exist several standard solvers, facilitating the implementation of the method. The drawback of single shooting is that the integrator call is a highly nonlinear operation, even if the differential equation is a linear one, making the NLP-optimizer prone to ending up in a local, rather than global, minimum and/or to slow convergence speeds. To overcome this, Bock's *direct multiple shooting* method [10] includes in the optimization problem the differential state at a number of points, "shooting nodes", and the continuity of the state trajectories at these points is enforced by adding additional constraints to the NLP. These additional degrees of freedom can be used for suitable initialization of the state trajectories and often increase the radius of convergence at the cost of a

larger NLP.

The main difficulty of the method, and often the bottleneck in terms of solution times, is to efficiently and accurately calculate first and often second order derivatives of the DAE integrator, needed by the NLP solver. Implementations of the method include MUSCOD-II and the open-source ACADO Toolkit [26], used here.

2.1.2 Direct collocation

A common alternative to shooting methods is direct collocation on finite elements. Direct collocation is a simultaneous method, where both the differential and algebraic states as well as the controls are approximated by polynomials. But in contrast to multiple shooting, no integrator is used to compute the state and algebraic profiles. Rather, the original continuous time optimal control problem (1) is transcribed directly into an algebraic problem in the form of a non-linear program (NLP). This non-linear program is usually large, but is also typically very sparse. Algorithms, such as IPOPT, [39], exist that explore the sparsity structure of the resulting NLP in order to compute solutions of the problem in a fast and robust way.

The interpolation polynomials used to approximate the state profiles are usually chosen to be orthogonal Lagrange polynomials, and common choices for the collocation points include Lobatto, Radau and Gauss schemes. In this paper, Radau collocation will be used, since this scheme has the advantage of featuring a collocation point at the end of each finite element, which makes encoding of continuity constraint for the states at element junction points straightforward.

The direct collocation method shares some characteristics with multiple shooting, since both are simultaneous methods. For example, unstable systems can be handled, and it is easy to incorporate state and control constraints. There are, however, also differences between multiple shooting and direct collocation. While a multiple shooting method typically requires computation of DAE sensitivities by means of integration of additional differential equations, direct collocation relies only on evaluation of first and (if analytic Hessian is used) second order derivatives of the DAE residual. For further discussion on the pros and cons of multiple shooting and direct collocation, see [9]

2.2 Modelica and Optimica

The Modelica language targets modeling of complex heterogeneous physical systems, [37]. Modelica

permits specification of models in a wide range of physical domains, including mechanics, thermodynamics, electronics, chemistry and thermal systems. Also, recent versions of the language support modeling of embedded control systems and mapping of controller code to real-time control hardware. Modelica is object-oriented and equation-based, where the former property provides a means to construct modular and reusable models and the latter enables the user to state declarative equations. It is worth noticing that both differential and algebraic equations are supported and that there is no need, for the user, to solve the model equations for the derivatives, which is common in block-based modeling frameworks. In addition, Modelica supports acausal modeling, enabling explicit modeling of physical interfaces. This feature is the foundation of the component model in Modelica, where components can be connected to each other in connection diagrams.

Whereas Modelica offers state-of-the-art modeling of complex physical systems, it lacks constructs for expressing optimization problems. For simulation applications, this is not a problem, but when integrating Modelica models with optimization frameworks, it is inconvenient. In order to improve the support for formulation of optimization problems, the Optimica extension [1] has been proposed. Optimica adds to Modelica a small number of constructs for expressing cost functions, constraints, and what parameters and controls to optimize.

2.3 JModelica.org

JModelica.org is a Modelica-based open source platform targeting optimization simulation and analysis of complex systems, [2]. The platform offers compilers for Modelica and Optimica, a simulation package called Assimulo and a direct collocation algorithm for solving large-scale DAE-based dynamic optimization problems. The user interface in JModelica.org is based on Python, which provides means to conveniently develop complex scripts and applications. In particular, the packages Numpy [30], Scipy [16] and Matplotlib [27] enable the user to perform numerical computations interactively.

Recent developments of the JModelica.org platform includes import and export of Functional Mock-up Units (FMUs) and the integration with ACADO Toolkit and CasADi reported in this paper. The JModelica.org platform has been used in several industrial applications, including [28, 5, 35, 31, 23, 11]

The JModelica.org compilers generate C-code in-

tended for compilation and linking with numerical solvers. While this is a well established procedure for compiling Modelica models, it suffers from some drawbacks. Compiled code indeed offers very efficient evaluation of the model equations, but it also requires the user to regard the model as a black box. In contrast, there are many algorithms that can make efficient use of models expressed in symbolic form. Examples include tools for control design, optimization algorithms, and code generation, see [12] for a detailed treatment of this topic. In order to offer an alternative format for model export, JModelica.org supports the XML format described in [32]. This format is an extension of the XML scheme specified by the Functional Mock-up Interface (FMI) specification [29] and contains, apart from model meta data also the model equations. The equations are given in a format that is closely related to the expression trees that are common in compilers. The XML export functionality is explored in this paper to integrate the packages ACADO Toolkit and CasADi with the JModelica.org platform.

2.4 ACADO Toolkit

ACADO Toolkit [26] is an open-source tool for automatic control and dynamic optimization developed at the Center of Excellence on Optimization in Engineering (OPTEC) at the K.U. Leuven, Belgium. It implements among other things Bock's direct multiple shooting method [10], and is in particular designed to be used efficiently in a closed loop setting for nonlinear model predictive control (NMPC). For this aim, it uses the *real-time iteration scheme*, [14], and solves the NLP by a structure exploiting sequential quadratic programming method using the active-set quadratic programming (QP) solver qpOASES, [18].

Compared to other tools for dynamic optimization, the focus of ACADO Toolkit has been to develop a complete toolchain, from the DAE integration to the solution of optimal control problems in realtime. This vertical integration, together with its implementation in self-contained C++ code, allows for the tool to be efficiently deployed on embedded systems for solving optimization-based control and estimation problems.

3 CasADi

CasADi is a minimalistic computer algebra system implementing automatic differentiation, AD (see [22]) in forward and adjoint modes by means of a hybrid symbolic/numeric approach, [4]. It is designed to be a low-

level tool for quick, yet highly efficient implementation of algorithms for numerical optimization, as illustrated in this paper, see Section 4. Of particular interest is dynamic optimization, using either a collocation approach, or a shooting-based approach using embedded ODE/DAE-integrators. In either case, CasADi relieves the user from the work of efficiently calculating the relevant derivative or ODE/DAE sensitivity information to an arbitrary degree, as needed by the NLP solver. This together with an interface to Python, see Section 3.1, drastically reduces the effort of implementing the methods compared to a pure C/C++/Fortran approach.

Whereas conventional AD tools are designed to be applied black-box to C or Fortran code, CasADi allows the user to build up symbolic representations of functions in the form of computational graphs, and then apply the automatic differentiation code to the graph directly. These graphs are allowed to be more general than those normally used in AD tools, including (sparse) matrix-valued operations, switches and integrator calls. To prevent that this added generality comes to the cost of lower numerical efficiency, CasADi also includes a second, more restricted graph formulation with only scalar, built-in unary and binary operations and no branches, similar to the ones found in conventional AD tools.

CasADi is an open source tool, written as a self-contained C++ code, relying only on the standard template library.

3.1 Python interface to CasADi

Whereas the C++ language is highly efficient for high performance calculations, and well suited for integration with numerical packages written in C or Fortran, it lacks the interactivity needed for rapid prototyping of new mathematical algorithms, or applications of an existing algorithm to a particular model. For this purpose, a scripting language such as Python, [36], or a numerical computing environment such as Matlab, is more suitable. We choose here to work with Python rather than Matlab due to its open source availability and ease of interfacing with other programming languages.

Interfacing C++ with Python can be done in several ways. One way is to wrap the C++ classes in C functions and blend them into a Python-to-C compiler such as Cython. While this approach is simple enough for small C++ classes, it becomes prohibitively cumbersome for more complex classes. Fortunately, there exist excellent tools that are able to automate

this process, such as the Simplified Wrapper and Interface Generator (SWIG) [17, 6] and the Boost-Python package. We have chosen to work with SWIG due to SWIG's support for a large subset of C++ constructs, a large and active development community and the possibility to interface the code to a variety of languages in addition to Python, in particular JAVA and Octave.

The latest version of SWIG at the time of writing, version 2.0, maps C++ language constructs onto equivalent Python constructs. Examples of features that are supported in SWIG are polymorphism, exceptions handling, templates and function overloading.

By carefully designing the CasADi C++ source code, it was possible to automatically generate interface code for all the public classes of CasADi. Since the interface code is automatically generated, it is easy to maintain as the work on CasADi progresses with new features being added. Using CasADi from Python renders little or no speed penalty, since virtually all work-intensive calculations (numerical calculation, operations on the computational graphs etc.), take place in the built-in virtual machine. Functions formulated in Python are typically called only once, to build up the graph of the functions, thereafter the speed penalty is negligible.

3.2 The CasADi interfaces to numerical software

In addition to being a modeling environment and an efficient AD environment, CasADi offers interfaces to a set of numeric software packages, in particular:

- The sensitivity capable ODE and DAE integrators CVODES and IDAS from the Sundials suite [25]
- The large-scale, primal-dual interior point NLP solver IPOPT [39]
- The ACADO toolkit

In the Sundials case, CasADi automatically formulates the forward or adjoint sensitivity equations and provides Jacobian information with the appropriate sparsity needed by the linear solvers, normally an involved and error prone process. For IPOPT, the gradient of the objective function is generated via adjoint AD. Also, a sparse Jacobian of the NLP constraints as well as an exact sparse Hessian of the Lagrangian can be generated using AD by source code transformation. The ACADO Toolkit interface makes it possible to use the tool from Python and attach an arbitrary ODE/DAE integrator (currently CVODES, IDAS

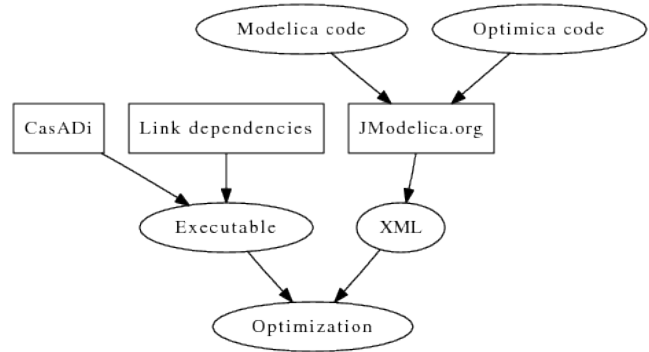


Figure 1: Optimization toolchain for JModelica.org/CasADi.

or fixed-step explicit integrators that have been implemented symbolically by the user) to ACADO.

3.3 Complete tool chain

Though models for relatively simple dynamic systems can be efficiently formulated directly in CasADi, for more complex models it is beneficial to use a more expressive approach based on an object-oriented modelling language such as Modelica. To transmit model information about the dynamic system between Modelica and CasADi, we use the XML exchange format reported in [32], which is supported by JModelica.org. On the CasADi side, an XML interpreter based on the open source XML parser TinyXML, [38], is used to parse the generated XML code and build up the corresponding C++ data structures. The complete toolchain is presented in Figure 1.

This approach contrasts to the more conventional approach currently used in the current optimization framework of JModelica.org. This approach is based on C-code generation, which then needs to be compiled by a C compiler and linked with JModelica.org's runtime environment. See Figure 2.

The fact that the approach does not rely on a C-compiler in the optimization loop means that the program code can be compiled and linked once and for all for a particular system and then distributed as executables. It is also important to note that as models grow in size, the time needed to compile the code may be large.

3.4 A simple example

To demonstrate the tool, we show how to implement a simple, single shooting method for the Van der Pol oscillator used as a benchmark in section 5.1:

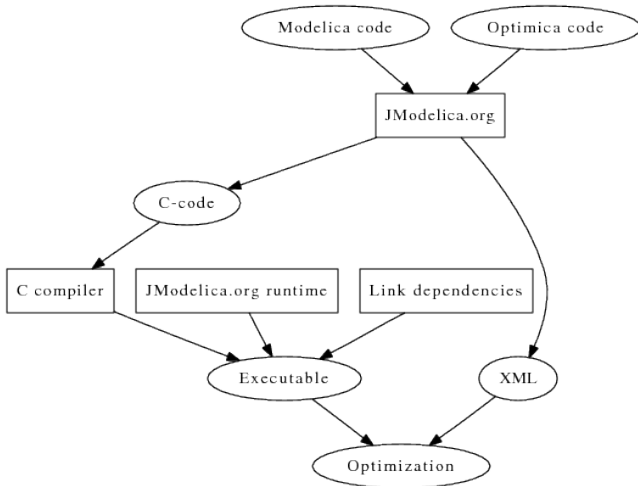


Figure 2: Optimization toolchain for JModelica.

```

from casadi import *

# Declare variables (use simple, efficient DAG)
t = SX("t") # time
x=SX("x"); y=SX("y"); u=SX("u"); L=SX("cost")

# ODE right hand side function
f = [(1 - y*y)*x - y + u, x, x*x + y*y + u*u]
rhs = SXFunction([[t],[x,y,L],[u]],[f])

# Create an integrator (CVODES)
I = CVodesIntegrator(rhs)
I.setOption("ad_order",1) # enable AD
I.setOption("abstol",1e-10) # abs. tolerance
I.setOption("reltol",1e-10) # rel. tolerance
I.setOption("steps_per_checkpoint",1000)
I.init()

# Number of control intervals
NU = 20

# All controls (use complex, general DAG)
U = MX("U",NU) # NU-by-1 matrix variable

# The initial state (x=0, y=1, L=0)
X = MX([0,1,0])

# Time horizon
T0 = MX(0); TF = MX(20.0/NU)

# State derivative and algebraic state
XP = MX(); Z = MX() # Not used

# Build up a graph of integrator calls
for k in range(NU):
    [X,XP,Z] = I.call([T0,TF,X,U[k],XP,Z])

# Objective function: L(T)
    
```

```

F = MXFunction([U],[X[2]])

# Terminal constraints: 0<=[x(T);y(T)]<=0
G = MXFunction([U],[X[0:2]])

solver = IpoptSolver(F,G)
solver.setOption("tol",1e-5)
solver.setOption("hessian_approximation", \
    "limited-memory")
solver.setOption("max_iter",1000)
solver.init()

# Set bounds and initial guess
solver.setInput(NU*[-0.75], NLP_LBX)
solver.setInput(NU*[1.0], NLP_UBX)
solver.setInput(NU*[0.0], NLP_X_INIT)
solver.setInput([0,0], NLP_LBG)
solver.setInput([0,0], NLP_UBG)

# Solve the problem
solver.solve()
    
```

In CasADi, symbolic variables are instances of either the scalar expression class SX, or the more general matrix expression class MX.

```

x=SX("x"); y=SX("y"); u=SX("u"); L=SX("cost")
...
U = MX("U",NU) # NU-by-1 matrix variable
    
```

In the example above, we declare variables and formulate the right-hand-side of the integrator symbolically:

```

f = [(1 - y*y)*x - y + u, x, x*x + y*y + u*u]
    
```

Note that at the place where this is encountered in the script, neither x , y or u have taken a particular value. This representation of the ordinary differential equation is passed to the ODE integrator CVODES from the Sundials suite [25]. Since the ODE is in symbolic form, the integrator interface is able to derive any information it might need to be able to solve the initial value problem efficiently, relieving the user of a tedious and often error prone process. The information that can be automatically generated includes derivative information for sparse, dense or banded methods, as well as the formulation of the forward and adjoint sensitivity equations (required here since the integrator is being used in an optimal control setting).

The next interesting line is:

```

for k in range(NU):
    [X,XP,Z] = I.call([T0,TF,X,U[k],XP,Z])
    
```

Here, the `call` member function of the `CVodesIntegrator` instance is used to construct a graph with function calls to the integrator. Since the

matrix variable U is not known at this point, actually solving the IVP is not possible. This allows us to get a completely symbolic representation not only of the ODE, but of the nonlinear programming program. We then pass the NLP to the open source dual-primal interior point NLP solver IPOPT. Again, since the formulation is symbolic, the IPOPT interface will generate all the information it needs to solve the problem, including the gradient of the NLP objective function and the Jacobian of the NLP constraint function, both of which can be best calculated using automatic differentiation in adjoint mode for this particular example.

Note that the example above, with comments removed, consists of about 30 lines of code, which is a very compact way to implement the single shooting method. With only moderately more effort, other methods from the field of optimal control can be formulated including multiple-shooting and direct collocation, see Section 4. When executing the script above, it iterates to the the correct solution in 592 NLP iterations, which is considerably slower than the corresponding results for the simultaneous methods. Adapting the script to implement multiple-shooting rather than single-shooting (the code of which is available in CasADi's example collection), decreases the number of NLP iterations to only 17.

4 A Python-based Collocation Algorithm

As described in Section 2.1, one strategy for solving large-scale dynamic optimization problems is direct collocation, where the dynamic DAE constraint is replaced by a discrete time approximation. The result is a non-linear program (NLP), which can be solved with standard algorithms. A particular challenge when implementing collocation algorithms is that the algorithms typically used to solve the resulting NLP require accurate derivative information and sparsity structures. In addition, second order derivatives can often improve robustness and convergence of such algorithms.

One option for implementing collocation algorithm is provided by optimization tools such as AMPL [19] and GAMS [21]. These tools support formulation of linear and non-linear programs and the user may specify collocation problems by encoding the model equations as well as the collocation constraints. The AMPL platform also provides a solver API, supporting evaluation of the cost function and the constraints, as well

as first and second order derivatives, including sparsity information. A benefit for the user is that the tool internally computes these quantities using an automatic differentiation strategy that is very efficient, which in turn enables a solver algorithm to operate fast and reliably. On the other hand, AMPL, and similar systems, does not offer appropriate support for physical modeling. The description format is inherently flat, which makes construction of reusable models intractable.

Physical modeling systems, on the other hand, offer excellent support for modeling and model reuse, but typically offer only model execution interfaces that often do not provide all the necessary API functions. Typically, sparsity information and second order derivative information is lacking. The model execution interface in JModelica.org, entitled the JModelica.org Model Interface (JMI) overcomes some of these deficiencies by providing a DAE interface supporting sparse Jacobians, which in turn are computed using the CppAD package [7]. Based on JMI, a direct collocation algorithm has been implemented in C and the resulting NLP has been interfaced with the algorithm IPOPT [39]. While this approach has been successfully used in a number of industrially relevant applications, it also requires a significant effort in terms of implementation and maintenance. In many respects, implementation of collocation algorithms reduces to book keeping problems where indices of states, inputs and parameters need to be tracked in the global variable vector. Also, the sparsity structure of the DAE Jacobian needs to be mapped into the composite NLP resulting from collocation. In this respect, the approach taken in AMPL and GAMS has significant advantages.

In an effort to explore the strengths of the physical modeling framework JModelica.org and the convenience and efficiency in evaluation of derivatives offered by CasADi, a direct collocation algorithm similar to the one existing in JModelica.org has been implemented. The implementation is done completely in Python relying on CasADi's model import feature and its Python interface. As compared to the approach taken with AMPL, the user is relieved from the burden of implementing the collocation algorithm itself. In this respect the new implementation does not differ from the current implementation in JModelica.org, but instead, the effort needed to implement the algorithm is significantly reduced. Also, advanced users may easily tailor the collocation algorithm to their specific needs.

The implementation used for the benchmarks presented in this paper is a third order Radau scheme,

which is also supported by the C collocation implementation in JModelica.org.

5 Benchmarks

Three different optimal control benchmark problems with different properties have been selected for comparison of the different algorithms: the Van der Pol oscillator, a Continuously Stirred Tank Reactor (CSTR) with an exothermic reaction, and a combined cycle power plant. The first benchmark, the Van der Pol oscillator, is a system commonly studied in non-linear control courses, and demonstrates the ability of all methods evaluated to solve optimal control problems. The CSTR problem features highly non-linear dynamics in combination with a state constraint. The final benchmark, the combined cycle power plant, is of larger scale, consisting of nine states and more than 100 algebraics.

Whereas the Van der Pol problem has been successfully solved using both multiple shooting and collocation, a solution to the CSTR and combined cycle problem has been obtained only using a collocation approach.

For reference, the original collocation implementation, written in C, is included in the benchmarks. This algorithm is referred to as JM collocation.

All the calculations have been performed on an Dell Latitude E6400 laptop with an Intel Core Duo processor of 2.4 GHz, 4 GB of RAM, 3072 KB of L2 Cache and 128 kB of L1 cache, running Linux.

In the benchmarks where IPOPT is used, the algorithm is compiled with the linear solver MA57 from the HSL suite.

5.1 Optimal control of the Van der Pol Oscillator

As a first example, consider the Van der Pol oscillator, described by the differential equations

$$\begin{aligned} \dot{x}_1 &= x_2, & x_1(0) &= 1 \\ \dot{x}_2 &= (1 - x_1^2)x_2 - x_1 & x_2(0) &= 0. \end{aligned} \quad (2)$$

The optimization problem is formulated as to minimize the following cost

$$\min_u \int_0^{20} x_1^2 + x_2^2 + u^2 dt \quad (3)$$

subject to the constraint

$$u \leq 0.75. \quad (4)$$

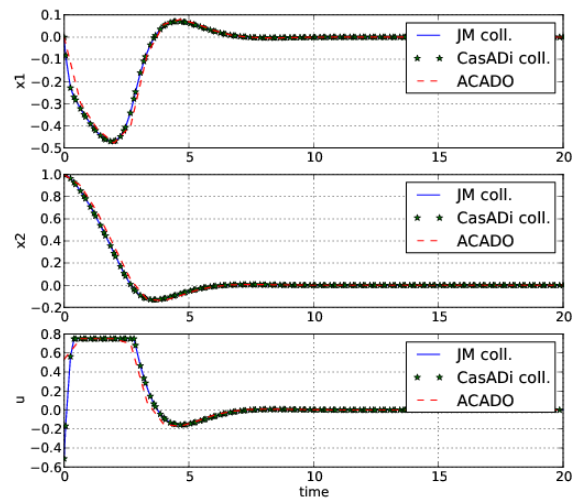


Figure 3: Optimization results for the Van der Pol oscillator.

First, we solve the optimal control problem using three different algorithms, all based on the Modelica model and the Optimica specification encoding the optimal control problem. 20 uniformly distributed control segments were used in all cases. The first method used to solve the problem is JModelica's native, C-based implementation of direct collocation, which relies on code generation and the AD-tool CppAD to generate derivatives. Secondly, the novel, CasADi and Python-based implementation of direct collocation presented in Section 4 is applied. The third algorithm is ACADO Toolkit's implementation of multiple shooting, using CasADi's interface to evaluate functions and directional derivatives. Forward mode AD was used to generate DAE sensitivities and a BFGS approximation of the Hessian of the Lagrangian of the NLP. The optimal solutions for the three different algorithms are shown in Figure 3.

Table 1 shows the number of NLP iterations and total CPU time for the optimization (in seconds) for 5 different algorithmic approaches: direct collocation implemented in C and in Python using CasADi, ACADO Toolkit via the CasADi interface, and implementations of single and multiple shooting based on CasADi's Python interface. The implementations of single and multiple shooting are included to demonstrate usage of CasADi's integrator and NLP solver implementation, rather than to achieve high performance. For the single shooting code, the complete script was presented in Section 3.4.

The last column of the table contains the share of the

total time spent in the NLP solver, the rest is mostly spent in the DAE functions (DAE residual, Jacobian of the constraints, objective function etc.) which are interfaced to the NLP solver. This information is not available for ACADO Toolkit.

Table 1: Execution times for the Van der Pol benchmark.

Tool	NLP iterations	Total time [s]	Time in NLP solver [s]
JModelica.org coll.	21	0.32	0.14
CasADi collocation	103	0.97	0.92
ACADO via CasADi	28	3.45	n/a
CasADi single shooting	616	167.7	1.22
CasADi mult. shooting	16	59.1	0.20

We can see that JModelica's current C-based implementation of direct collocation and ACADO's multiple shooting implementation, both of them using an inexact Hessian approximation with BFGS updating, show a similar number of NLP iterations. In terms of speed, the JModelica.org implementation clearly outperforms ACADO which is at least partly explained by the fact that JModelica.org involves a code generation step, significantly reducing the function overhead in the NLP solver. Also, CasADi involves a code generation step, but not to C-code which is then compiled, but to a virtual machine implemented inside CasADi. Clearly, this virtual machine is able to compete with the C implementation in terms of efficiency during evaluation and is also fast in the translation phase, as no C-code needs to be generated and compiled. Also note that for this small problem size, the function overhead associated with calling the DAE right hand side is major for all of the shooting methods. A more fair comparison here would involve the use of ACADO Toolkit's own symbolic syntax coupled with a code generation step in ACADO.

In this particular example, IPOPT, using CasADi to generate the exact Hessian of the Lagrangian, require more NLP steps than ACADO and JModelica.org's native implementation of collocation which are both using an inexact Hessian approximation. Despite the fact that the CasADi-based implementation does not rely on C code generation, and although it is calculating the exact Hessian, the time it takes to evaluate the functions only constitute only a small fraction (around 5%) of the total execution time. In contrast, in the C-based implementation, the time spent in DAE functions, make up more than half of the total CPU time. This result clearly demonstrates one of the main strengths of

CasADi, namely computational efficiency.

Looking at the number of iterations required in the single shooting algorithm, the superior convergence speed of simultaneous methods (collocation and multiple shooting) is obvious.

5.2 Optimal control of a CSTR reactor

We consider the Hicks-Ray Continuously Stirred Tank Reactor (CSTR) containing an exothermic reaction, [24]. The states of the system are the reactor temperature T and the reactant concentration c . The reactant inflow rate, F_0 , concentration, c_0 , and temperature, T_0 , are assumed to be constant. The input of the system is the cooling flow temperature T_c . The dynamics of the system is then given by:

$$\begin{aligned}\dot{c}(t) &= F_0(c_0 - c(t))/V - k_0 e^{-E_{divR}/T(t)} c(t) \\ \dot{T}(t) &= F_0(T_0 - T(t))/V - \\ & dH/(\rho C_p) k_0 e^{-E_{divR}/T(t)} c(t) + \\ & 2U/(r\rho C_p)(T_c(t) - T(t))\end{aligned}\quad (5)$$

where r , k_0 , E_{divR} , U , ρ , C_p , dH , and V are physical parameters.

Based on the CSTR model, the following dynamic optimization problem is formulated:

$$\min_{T_c(t)} \int_0^{t_f} (c^{\text{ref}} - c(t))^2 + (T^{\text{ref}} - T(t))^2 + (T_c^{\text{ref}} - T_c(t))^2 dt \quad (6)$$

subject to the dynamics (5). The cost function corresponds to a load change of the system and penalizes deviations from a desired operating point given by target values c^{ref} , T^{ref} and T_c^{ref} for c , T and T_c respectively. Stationary operating conditions were computed based on constant cooling temperatures $T_c = 250$ (initial conditions) and $T_c = 280$ (reference point).

In order to avoid too high temperatures during the ignition phase of the reactor, the following temperature bound was enforced:

$$T(t) \leq 350. \quad (7)$$

The optimal trajectories for the three different algorithms are shown in Figure 4, where we have used a control discretization of 100 elements and a 3rd order Radau-discretization of the state trajectories for the two collocation implementations.

Table 2 shows the performance of the two compared implementations of collocation in terms of NLP iterations and CPU time.

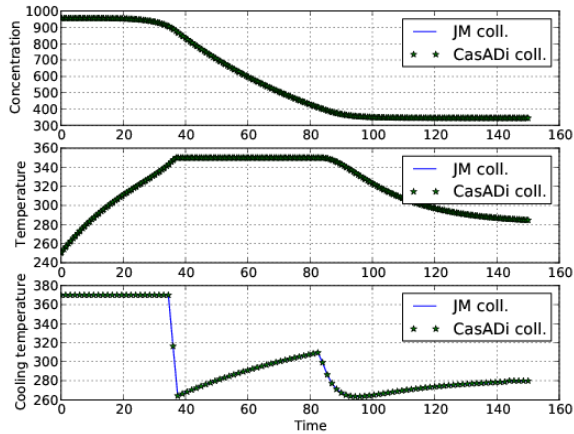


Figure 4: Optimization results for the CSTR reactor.

Table 2: Execution times for the CSTR benchmark.

Tool	JM coll.	CasADi coll.
NLP iterations	85	20
Total time	3.2	0.18
Time in NLP solver	1.1	0.16
Time in DAE functions	2.1	0.02

In this example, the Python-based collocation algorithm is clearly superior, it converges more quickly, which is likely due to the provided exact Hessian. Also, the lion's share of the execution time is spent internally in IPOPT, leaving little opportunities to optimize the code in function evaluations further.

5.3 Optimal start-up of a combined cycle power plant

5.3.1 Physical model setup

A simplified model of a one-level-of-pressure combined-cycle power plant is considered in this benchmark, see Figure 5 for the object diagram.

The gas turbine model (lower left) generates a prescribed flow of exhaust gas at a prescribed temperature, which are both a function of the load input signal.

The turbine exhaust gases enter the hot side of a counter-current heat exchanger, and are then discharged to the atmosphere. The economizer and superheater are modelled by a dynamic energy balance equation for each side, neglecting compressibility and friction effects. The drum boiler evaporator instead includes both dynamic mass and energy balance equations, assuming thermodynamic equilibrium between the liquid and the vapour phases. The energy storage

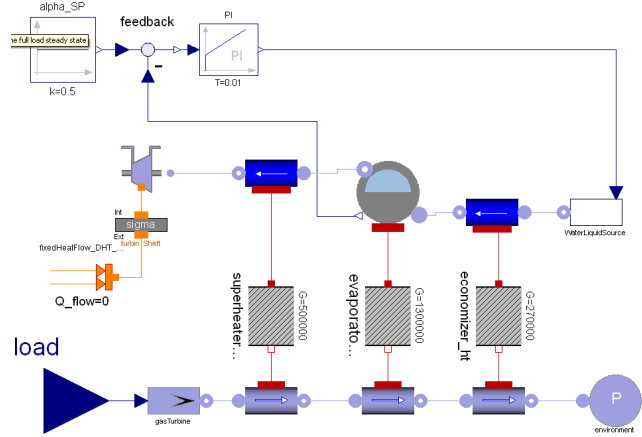


Figure 5: Diagram of the combined-cycle plant model.

in all the steel walls is accounted for assuming they are at the same temperature of the water/steam fluid.

The feedwater system is described by a prescribed flow rate source with fixed temperature, driven by a PI level controller that stabilizes the level dynamics and keeps the void fraction in the drum around 0.5.

Finally, the superheated steam enters the steam turbine, which is modeled as an expansion to the condenser pressure, assuming a constant isentropic efficiency. The turbine also exposes a thermal port, corresponding to the surface where the inlet steam comes into contact with the turbine rotor. This port is connected to a thermal model of the hollow shaft, given by Fourier's heat equation, discretized by the finite difference method. The thermal stress on the shaft surface, which is the main limiting factor in the start-up transients, is proportional to the difference between the surface and the average temperature of the shaft.

In order to keep the complexity low, constant specific heat c_p is assumed in the economizers and superheaters; lumped-parameter models are assumed for the heat exchanger segments, with just one temperature state for each side. Last, but not least, also the turbine rotor thermal model has only one temperature state resulting from the discretization of Fourier's equation. The resulting nonlinear model has nine state variables and 127 algebraic variables. A more detailed discussion on the physical modelling of the plant can be found in [11].

5.3.2 Minimum-time start-up

The goal of the optimization problem is to reach the full load level as fast as possible, while limiting the peak stress value on the rotor surface, which deter-

mines the lifetime consumption of the turbine. Since the steam cycle is assumed to operate in a pure sliding pressure mode, the full load state is reached when the load level of the turbine, $u(t)$ (which is the control variable), has reached 100% and the normalized value of the evaporator pressure, p_{ev} , has reached the target reference value p_{ev}^{ref} . A Lagrange-type cost function, penalizing the sum of the squared deviations from the target values, drives the system towards the desired set-point $(p_{evap}, u) = (p_{evap}^{ref}, 1)$ as quickly as possible.

Inequality constraints are prescribed on the maximum admissible thermal stress in the steam turbine, $\sigma(t)$, as well as on the rate of change of the gas turbine load: on one hand, the load is forbidden to decrease, in order to avoid cycling of the stress level during the transient; on the other hand, it cannot exceed the maximum rate prescribed by the manufacturer.

The start-up optimization problem is then defined as:

$$\min_{u(t)} \int_{t_0}^{t_f} (p_{evap}(t) - p_{evap}^{ref})^2 + (u(t) - 1)^2 dt \quad (8)$$

subject to the constraints

$$\begin{aligned} \sigma(t) &\leq \sigma_{max} \\ \dot{u}(t) &\leq du_{min} \\ \dot{u}(t) &\geq 0 \end{aligned} \quad (9)$$

and to the DAE dynamics representing the plant. The initial state for the DAE represents the state of the plant immediately after the steam turbine roll-out phase and the connection of the electric generator to the grid.

The optimization result is shown in Figure 6. During the first 200 seconds, the gas turbine load is increased at the maximum allowed rate and the stress builds up rapidly, until it reaches the target limit. Subsequently, the load is slowly increased, in order to maintain the stress level approximately constant at the prescribed limit. When the 50% load level is reached, further increases of the load do not cause additional increase of the gas exhaust temperature, and therefore cause only small increases of the steam temperature. It is then possible to resume increasing the load at the maximum allowed rate, while the stress level starts to decrease. The full load is reached at about 1400 s. Between 1000 and 1100 seconds, the load increase rate is actually zero; apparently, this small pause allows to increase the load faster later on, leading to an overall shorter start-up time.

This problem, which is of a more realistic size has been solved with the two direct collocation implementations and the results are shown in Table 3.

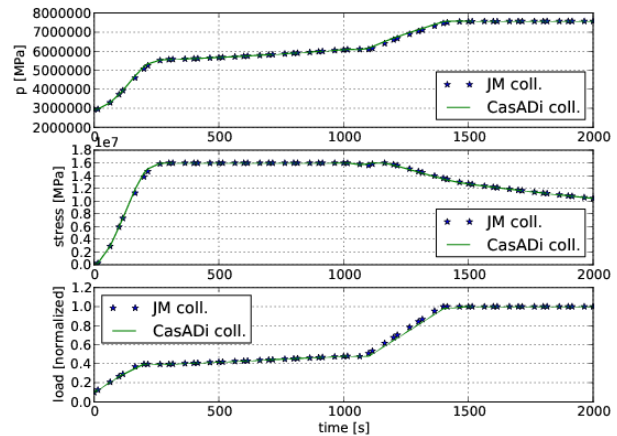


Figure 6: Optimal startup of a combined cycle power plant.

Table 3: Execution times for the combined cycle benchmark.

Tool	JM coll	CasADi coll
NLP iterations	49	198
Total time	19.9	16.3
Time in NLP solver	2.4	15.3
Optimal cost	6487	6175

We note that the CasADi-based collocation algorithm needs more iterations to reach the optimal solution, but on the other hand, the optimum that it found is indeed better than the one found using BFGS. Whether this is due to some minor differences in the collocation algorithms (since the models are identical), is beyond the scope of this paper. What can be said with certainty is that whereas most of the time spent in the DAE functions in the existing C-based approach, the opposite is true for the CasADi approach. Indeed, with more than 90% of the computational time spent internally in IPOPT, optimizing the CasADi execution time further would do little to reduce the overall execution time.

6 Summary and Conclusions

In this paper, the integration of CasADi and the JModelica.org platform has been reported. It has been shown how an XML-based model exchange format supported by JModelica.org and CasADi is used to combine the expressive power provided by Modelica and Optimica with state of the art optimization algorithms. The use of a language neutral model exchange format simplifies tool interoperability and allows users

to use different optimization algorithms without the need to reencode the problem formulation. As compared to traditional optimization frameworks, typically requiring user's to encode the model, the cost function and the constraints in a algorithm-specific manner, the approach put forward in this paper increases flexibility significantly.

7 Acknowledgments

This research at KU Leuven was supported by the Research Council KUL via the Center of Excellence on Optimization in Engineering EF/05/006 (OPTEC, <http://www.kuleuven.be/optec/>), GOA AMBioRICS, IOF-SCORES4CHEM and PhD/postdoc/fellow grants, the Flemish Government via FWO (PhD/postdoc grants, projects G.0452.04, G.0499.04, G.0211.05, G.0226.06, G.0321.06, G.0302.07, G.0320.08, G.0558.08, G.0557.08, research communities ICCoS, ANMMM, MLDM) and via IWT (PhD Grants, McKnow-E, Eureka-Flite+), Helmholtz Gemeinschaft via vICeRP, the EU via ERNSI, Contract Research AMINAL, as well as the Belgian Federal Science Policy Office: IUAP P6/04 (DYSCO, Dynamical systems, control and optimization, 2007-2011).

Johan Åkesson gratefully acknowledges financial support from the Swedish Science Foundation through the grant *Lund Center for Control of Complex Engineering Systems (LCCC)*.

References

- [1] Johan Åkesson. Optimica—an extension of modelica supporting dynamic optimization. In *In 6th International Modelica Conference 2008*. Modelica Association, March 2008.
- [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 problem. *Computers and Chemical Engineering*, 34(11):1737–1749, November 2010. Doi:10.1016/j.compchemeng.2009.11.011.
- [3] Johan Åkesson and Ola Slätteke. Modeling, calibration and control of a paper machine dryer section. In *5th International Modelica Conference 2006*, Vienna, Austria, September 2006. Modelica Association.
- [4] J. Andersson, B. Houska, and M. Diehl. Towards a Computer Algebra System with Automatic Differentiation for use with Object-Oriented modelling languages. In *3rd International Workshop on Equation-Based Object-Oriented Modelling Languages and Tools, Oslo, Norway, October 3, 2010*.
- [5] Niklas Andersson, Per-Ola Larsson, Johan Åkesson, Staffan Haugwitz, and Bernt Nilsson. Calibration of a polyethylene plant for grade change optimizations. In *21st European Symposium on Computer-Aided Process Engineering*, May 2011. Accepted for publication.
- [6] D. M. Beazley. Automated scientific software scripting with SWIG. *Future Gener. Comput. Syst.*, 19:599–609, July 2003.
- [7] B. M. Bell. CppAD Home Page, 2010. <http://www.coin-or.org/CppAD/>.
- [8] Lorenz T. Biegler. *Nonlinear programming: concepts, algorithms, and applications to chemical processes*. SIAM, 2010.
- [9] 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.
- [10] 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.
- [11] F. Casella, F. Donida, and J. Åkesson. Object-oriented modeling and optimal control: A case study in power plant start-up. In *Proc. 18th IFAC World Congress*, 2011. In submission.
- [12] Francesco Casella, Filippo Donida, and Johan Åkesson. An XML representation of DAE systems obtained from Modelica models. In *Proceedings of the 7th International Modelica Conference 2009*. Modelica Association, September 2009.
- [13] Dassault Systemes. Dymola web page, 2010. <http://www.3ds.com/products/catia/portfolio/dymola>.

- [14] 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. *J. Proc. Contr.*, 12(4):577–585, 2002.
- [15] M. Diehl, H. J. Ferreau, and N. Haverbeke. *Nonlinear model predictive control*, volume 384 of *Lecture Notes in Control and Information Sciences*, chapter Efficient Numerical Methods for Nonlinear MPC and Moving Horizon Estimation, pages 391–417. Springer, 2009.
- [16] Inc. Enthought. SciPy, 2010. <http://www.scipy.org/>.
- [17] Dave Beazley et al. SWIG – Simplified Wrapper and Interface Generator, version 2.0, 2010. <http://www.swig.org/>.
- [18] H. J. Ferreau, H. G. Bock, and M. Diehl. An online active set strategy to overcome the limitations of explicit MPC. *International Journal of Robust and Nonlinear Control*, 18(8):816–830, 2008.
- [19] R Fourer, D. Gay, and B Kernighan. *AMPL – A Modeling Language for Mathematical Programming*. Brooks/Cole — Thomson Learning, 2003.
- [20] R. Franke, M. Rode, and K Krüger. On-line optimization of drum boiler startup. In *Proceedings of Modelica’2003 conference*, 2003.
- [21] Gams Development Corporation. GAMS web page, 2010. <http://www.gams.com/>.
- [22] A. Griewank. *Evaluating Derivatives, Principles and Techniques of Algorithmic Differentiation*. Number 19 in Frontiers in Appl. Math. SIAM, Philadelphia, 2000.
- [23] Staffan Haugwitz, Johan Åkesson, and Per Hagander. Dynamic start-up optimization of a plate reactor with uncertainties. *Journal of Process Control*, 19(4):686–700, April 2009.
- [24] G. A. Hicks and W. H. Ray. Approximation methods for optimal control synthesis. *Can. J. Chem. Eng.*, 20:522–529, 1971.
- [25] A. C. Hindmarsh, P. N. Brown, K. E. Grant, S. L. Lee, R. Serban, D. E. Shumaker, and C. S. Woodward. Sundials: Suite of nonlinear and differential/algebraic equation solvers. *ACM Transactions on Mathematical Software*, 31:363–396, 2005.
- [26] B. Houska, H.J. Ferreau, and M. Diehl. ACADO Toolkit – An Open Source Framework for Automatic Control and Dynamic Optimization. *Optimal Control Applications and Methods*, 2010. DOI: 10.1002/oca.939 (in print).
- [27] J. Hunter, D. Dale, and M. Droettboom. Matplotlib: Python plotting, 2010. <http://matplotlib.sourceforge.net/>.
- [28] P-O. Larsson, J. Åkesson, Staffan Haugwitz, and Niklas Andersson. Modeling and optimization of grade changes for multistage polyethylene reactors. In *Proc. 18th IFAC World Congress*, 2011. In submission.
- [29] Modelisar. Functional Mock-up Interface for Model Exchange, 2010. <http://www.functional-mockup-interface.org>.
- [30] T. Oliphant. Numpy Home Page, 2009. <http://numpy.scipy.org/>.
- [31] B. Olofsson, H. Nilsson, A. Robertsson, and J. Åkesson. Optimal tracking and identification of paths for industrial robots. In *Proc. 18th IFAC World Congress*, 2011. In submission.
- [32] Roberto Parrotto, Johan Åkesson, and Francesco Casella. An XML representation of DAE systems obtained from continuous-time Modelica models. In *Third International Workshop on Equation-based Object-oriented Modeling Languages and Tools - EOOLT 2010*, September 2010.
- [33] J. Poland, A. J. Isaksson, and P. Aronsson. Building and solving nonlinear optimal control and estimation problems. In *7th International Modelica Conference*, 2009.
- [34] Process Systems Enterprise. gPROMS Home Page, 2010. <http://www.psenterprise.com/gproms/index.html>.
- [35] K. Prölss, H. Tummescheit, S. Velut, Y. Zhu, J. Åkesson, and C. D. Laird. Models for a post-combustion absorption unit for use in simulation, optimization and in a non-linear model predictive control scheme. In *8th International Modelica Conference*, 2011.
- [36] Python Software Foundation. Python Programming Language – Official Website, January 2011. <http://www.python.org/>.

- [37] The Modelica Association. Modelica – a unified object-oriented language for physical systems modeling, language specification, version 3.2. Technical report, Modelica Association, 2010.
- [38] Lee Thomason. TinyXML, January 2011. <http://www.grinninglizard.com/tinyxml/>.
- [39] Andreas Wächter and Lorenz T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1):25–58, 2006.
- [40] V. M. Zavala and L. T. Biegler. The advanced step NMPC controller: Optimality, stability and robustness. *Automatica*, 45(1):86–93, 2009.

Towards a Modular and Accessible Modelica Compiler Backend

Jens Frenkel⁺, Günter Kunze⁺, Peter Fritzson^{*}, Martin Sjölund^{*}, Adrian Pop^{*}, Willi Braun[#]
⁺Dresden University of Technology, Institute of Mobile Machinery and Processing Machines

^{*}PELAB – Programming Environment Lab, Dept. Computer Science
 Linköping University, SE-581 83 Linköping, Sweden

[#]FH Bielefeld, University of Applied Sciences

{jens.frenkel, guenter.kunze}@tu-dresden.de, {peter.fritzson, martin.sjolund, adrian.pop}@liu.se,
 willi.braun@fh-bielefeld.de

Abstract

Modelica is well suited for modelling complex physical systems due to the acausal description it is using. The causalisation of the model is carried out prior to each simulation. A significant part of the causalisation process is the symbolic manipulation and optimisation of the model. Despite the growing interest in Modelica, the capabilities of symbolic manipulation and optimisation are not fully utilized. This paper presents an approach to increase the customisability, access, and reuse of symbolic optimisation by a more modular and flexible design concept. An overview of the common symbolic manipulation and optimisation algorithms of a typical Modelica compiler is presented as well as a general modular design concept for a Modelica compiler backend. The modularisation concept will be implemented in a future version of the OpenModelica compiler.

Keywords: Compiler Backend; Optimisation; Interfaces.

1 Introduction

Modelica is a multi-domain object-oriented equation based modelling language. It is mostly used for modelling and simulation of complex physical systems but other tasks like symbolic analysis of the equation system behind the model are possible as well. To accomplish those tasks a Modelica Compiler or Interpreter is needed. All Modelica compilers known to the authors are divided into a frontend and a backend. The frontend is used to extract the system of equations behind the object-oriented Modelica Model. This task is typically called flattening and forms a challenge on its own [1].

The Backend is the part of the compiler which performs symbolic manipulation of the equation systems and generates code suitable for (efficient) execution or interpretation. These symbolic manipulations are de-

pendent on the specific analysis task and the type of the equation system. To prepare a model for numerically stable simulation of a higher index system, for example, the backend could perform an index reduction in combination with a matching algorithm to assign all the equations and perform some symbolic optimisation in order to improve the runtime performance.

Because of the complexity of Modelica the typically implemented symbolic manipulation algorithms are generally applicable for all kinds of equation systems. On the one hand this is an advantage because the user does not need to deal with the tasks of symbolic manipulation to get the desired results. On the other hand it is a disadvantage for the advanced user or library developer because there is normally only restricted access to the symbolic manipulation algorithms.

To overcome these limitations the authors designed a concept for a modular Modelica compiler backend which to a large extent has been realised in the new reorganised implementation of the Open Source OpenModelica Compiler from OpenModelica version 1.6, excluding external APIs and phase reordering flexibility. The concept will include an interface for external tools, for example for symbolic optimisation, and a safe and task-dependent method to implement manipulation algorithms for equation systems.

The next chapter introduces the reader into the common symbolic manipulation process for Modelica Compilers. Chapter three presents an overview of advanced symbolic manipulation algorithms which may speed up the simulation. The options for the user to access and customisation of symbolic manipulation algorithms are discussed within chapter four.

After concluding the previous chapters, chapter five presents the concept of the new modular compiler backend. Chapter six continues with the basic concept for its implementation and chapter seven presents interfaces of the modular compiler backend in more detail.

2 Common Symbolic Manipulation in Modelica Compiler Backends

Simulation itself is the most common usage of Modelica Models. The following section gives a general overview on the tasks of the backend of the Modelica compiler and applies to almost all implementations, see also Figure 1.

The output of the frontend is usually a flat representation of the Modelica model containing all variables with their properties, equations and algorithms. The first step of the backend is to analyse the flat model representation and extract additional information. Extracted information could for example be the candidates for states, implicit discrete variables and the variables with time-dependency. Candidates for states are differentiated variables and variables with the property `stateSelect` equal to `StateSelect.prefer` or `StateSelect.always`.

With the change of Modelica 3.2 the derivative operator “`der`” may have an expression as its argument. With the ambition of an efficient code generation all such expressions have to be symbolically differentiated with respect to the time dependent variables. Only if it is not possible to perform the symbolic differentiation an additional variable should be introduced.

The next steps of the symbolic optimisation are to remove simple equations and equations with no time dependency. Simple equations are of the form “`a=b`”, “`a=-b`” also called “alias equations”. Alias equations appear quite often because of the connect-equations concept. Equations with no time dependency are composed of constant or parameter variables and variables calculated based on constants or parameters.

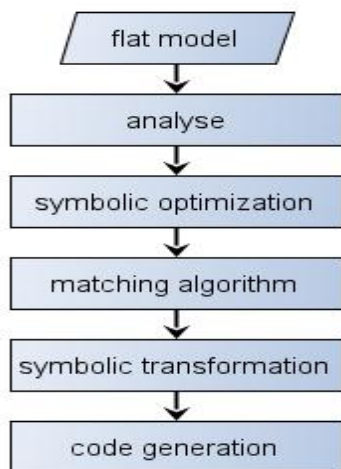


Figure 1: Common symbolic manipulation process for Modelica models.

Subsequently follows the matching algorithm which is one of the main phases of the backend. This algorithm has to find an equation for each variable the equation is solved for. If the system should be solved as an explicit ordinary differential equation (ODE) system, but is a higher index system, the matching algorithm works together with an index reduction algorithm.

To get the block lower triangular form (BLT-Form) Tarjan’s algorithm [10] is performed. This algorithm sorts the equations in the order they have to be solved.

The next step is the symbolic transformation and the collection of additional information, for example the occurring of zero crossings. Within the symbolic transformation phase the information from the matching algorithm is used to solve certain equations symbolically. This has to be done for single equations as well as for linear or nonlinear equation systems. To solve certain equations symbolically means to solve an equation or a system of equations for the desired variable or variables or provide all information to use an applicable numerical solver.

The last part of the backend after the whole symbolic manipulation process is concerned with code generation.

3 Advanced User Supported Symbolic Manipulation

In addition to the methods mentioned in Section 2 there are specialised symbolic manipulation algorithms which may speed up the simulation. These include algorithms like:

- dummy derivative with dynamic state selection
- tearing
- inline integration
- function inlining.

All to the author’s known Modelica compilers do not use these algorithms automatically, apart from tearing. Normally the users have to set special compiler flags or adjustments or use specific keywords, for example annotations for code generation.

The Dummy Derivative method [3] is a commonly used index reduction algorithm for higher index differential algebraic equations (DAE) systems. Using the Dummy Derivatives method may for some systems result in a singular Jacobian. A well-known example is the description of a planar pendulum described using the Cartesian coordinates.

To overcome these mathematical difficulties dynamic state selection could be used. The basic idea of

dynamic state selection is to change the section of state variables at run-time for higher performance and numerically stable simulation, see [4] and [5].

Large algebraic equation systems are typical of Modelica models. The tearing algorithm is a method to reduce the size of large algebraic equation systems by dividing strongly connected components in the BLT graph into smaller sub-systems of equations which could be solved more efficiently.

The basic concept of tearing is to make an assumption about one or several variables to be known. With this assumption the matching algorithm is continued. The torn equation systems are divided into several reduced algebraic equation systems and a set of explicit assignments. In case of a linear algebraic equation system the relaxation algorithm or a solver for nonlinear systems, for example the Newton Iteration method, can be used to solve the algebraic system. Detailed information on the tearing algorithm can be found in Cellier and Kofmann [6].

For simulations with strong requirements on execution time like hardware in the loop simulation, it may be beneficial to merge the integration method and the equation system. This method is called inline integration and yields an equation system where the numerical solver is eliminated as an explicit software component. Inline integration is for minimising call overhead and enabling additional symbolic manipulations for example removing the division of the time step on both sides of an equation. For further information about inline integration [6] and [7] are recommended.

The Modelica Specification 3.2 provides within chapter 17.3 annotations for code generation. The annotations:

- Evaluate,
- Inline and
- LateInline

are useful for symbolic optimisation. For example the Modelica MultiBody Library uses these annotations.

Performing the specialised symbolic manipulation algorithms the common symbolic manipulation process from Figure 1 is expanded to the advanced symbolic manipulation process shown in Figure 2. For some optimisation methods, for example inline integration, it is useful to run the matching and sorting algorithm repeatedly. Hence the graph presented in Figure 2 has in this case a loop within the second optimisation and matching algorithm stage.

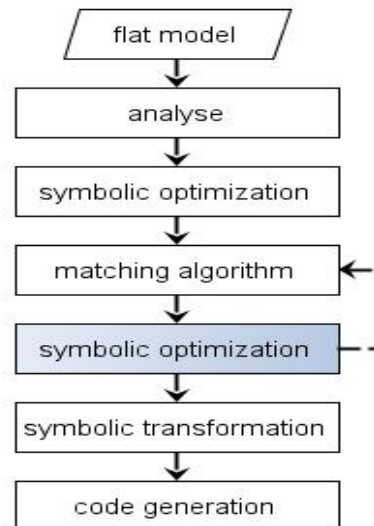


Figure 2: Advanced symbolic manipulation process for Modelica models.

4 Access to and Customisation of Symbolic Manipulation

Letting the user or library developers provide the compiler with specific information on how the symbolic manipulation algorithms should optimise the equation system can be achieved through several options:

1. Usage of specific keywords introduced into the language
2. Usage of optional compiler flags/settings
3. Exporting the systems of equations in a symbolic standardized and easy to use format and perform the optimisation with another tool
4. Development of a new Modelica compiler

However, none of the above mentioned options are practical in case a special optimisation is needed. The reasons are:

- Option 1 requires a long term standardisation process for the new methods.
- Option 2 is tool-specific, and therewith restricted.
- For Option 3 there is no standardized and easy to use format available yet nor does the common compiler support an export or import of the system of equations. Even if there is a support for export available the user would have to provide a code writer for the target format.
- Option 4 is not possible for most of the users.

In case an advanced user requires a symbolic manipulation algorithm, the compiler has to fulfill the following requirements:

- The export and import of the system of equations in a symbolic standardised and easy to use format. (A promising development is shown in [2].)
- An increasing number of symbolic manipulation algorithms.
- The possibility to define and use a symbolic manipulation algorithm in a safe and application-related way.

5 Modular Compiler Backend

To achieve the requirements of a more open and customisable access to symbolic manipulation capabilities the backend of the compiler needs a more general design concept. Keeping the tasks presented in Section 2 in mind the main tasks of the backend can be summarized into five phases:

- input phase
- pre-optimisation phase
- transformation phase
- post-optimisation phase
- output phase

Based on these phases and the concept of transferring the system of equations represented by a data object from one phase to the next, an equation system pipeline is formed as shown in Figure 3.

The transformation phase achieves a matching using index reduction methods and sorting the equations. Different modules for index reduction should be available as shown in Figure 3 within the left upper inner box “DAE-Handler for Index Reduction”.

For some optimisation methods, for example inline integration, it is useful to run the matching and sorting algorithm repeatedly. Hence the pipeline presented in Figure 3 has a loop within the post-optimisation and transformation phase.

The output phase transforms the system of equations into the desired target format. This phase is the combination from the symbolic transformation and the code generation from Figure 2. Supposable target formats are a simulation executable or a functional mockup unit or if not a simulation is desired a XML, Matlab or Python file export.

With a modularisation concept in mind each phase should be presented by one (input phase, transformation phase and output phase) or several (pre- and post-optimisation phases) modules. For a specific equation system the pipeline might be built from specific modules in a specific order. For another equation system the pipeline might be built from other modules in a different order. More precisely, it will be possible to use application-oriented pipelines for different equation

system. As mentioned above a Modelica model is represented in the backend as an equation system. This equation system could be composed of equations, algorithms and sub-systems of equations. In case of a multi-domain model it may be beneficial to use different optimisation modules for the different sub-systems present in the model. The equation system pipeline framework will allow the optimisation of different sub-systems using different optimisation modules. The challenge is to define proper rules for the differentiation of systems of equations.

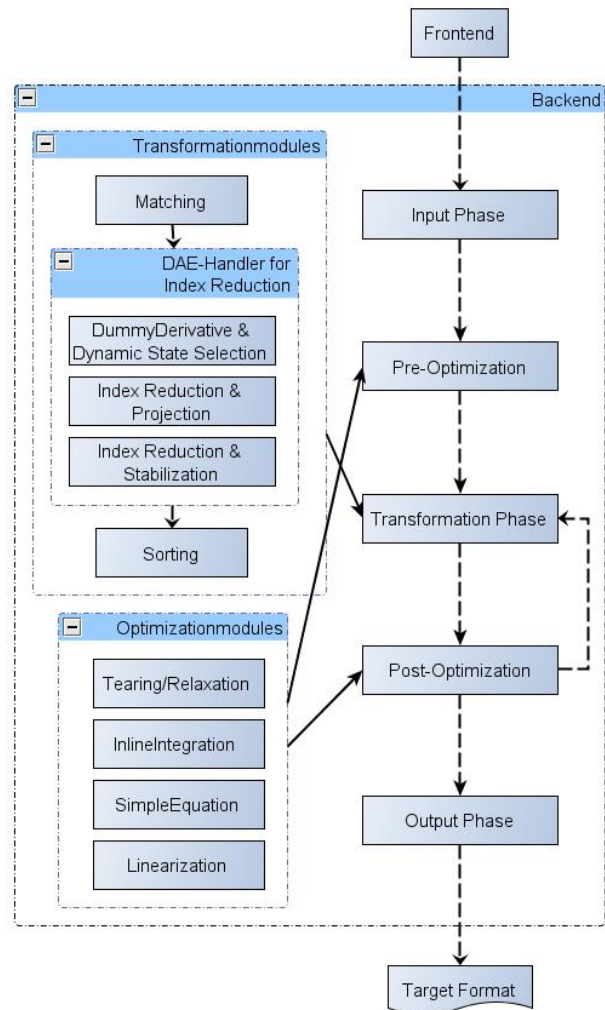


Figure 3: Data Flow in the Equation System Pipeline.

The system of equations has to be represented by a data object. This data object should basically include:

- the variables,
- the equations and
- the algorithms.

In case of an efficient implementation and to use the same interface for the pre- and the post-optimisation modules as desired, it is beneficial to store additional information within the data object and to classify varia-

bles, equations and algorithms into different groups. The additional information should be:

- an incidence matrix for the equations/algorithms and variables,
- the matching of variables and equations/algorithms solved for and
- the order of the equations/algorithms they have to be calculated in.

6 Concept for Implementation

To reduce the possibilities of introducing errors in developing the backend and support the developers in the most suitable way, three levels of complexity will be introduced by a class concept represented in Figure 4.

The basic level is a library for manipulating symbols and symbolic expressions, as shown in the innermost box called “Symbolic Math Library” of Figure 4. The Symbolic Math Library comprises the four modules:

- Expression
- Symbol
- Simplify
- Solve

An expression consists of symbols, numbers and operators. In case of a non-scalar symbol an expression is used to point to the scalar elements of the symbol. Hence, the expression module and the symbol module use one another as shown in Figure 4. The two modules comprise functions to:

- generate expressions/symbols,
- transform them to other types,
- manipulate ,
- get something,
- traverse,
- compare and
- replace sub-expressions.

The module “Simplify” performs symbolic simplification of expressions and the module “Solve” implements functions to solve symbolical equations with the form “0=exp” for sub-expression.

Based on this symbolic math library an equation system library presents the second level, which supports all operations to access and manipulate equations, algorithms and equation systems. The “Equation System Library” comprises the four modules:

- Variable
- Equation
- Algorithm
- EquationSystem

This library provides functions to manipulate the system of equations, traverse them or replace variables

with other variables or expressions for example. Both libraries have to preserve the consistency of the system of equations.

By using the library for symbols and expressions as well as the equation system library specific modules can be developed for each phase of the equation system pipeline within the third level in a safe and task-oriented way. The upper box “Modules for Optimisation” in Figure 4 for example represents optimisation algorithms such as:

- Tearing/Relaxation
- InlineIntegration
- Linearisation
- Remove Simple Equation

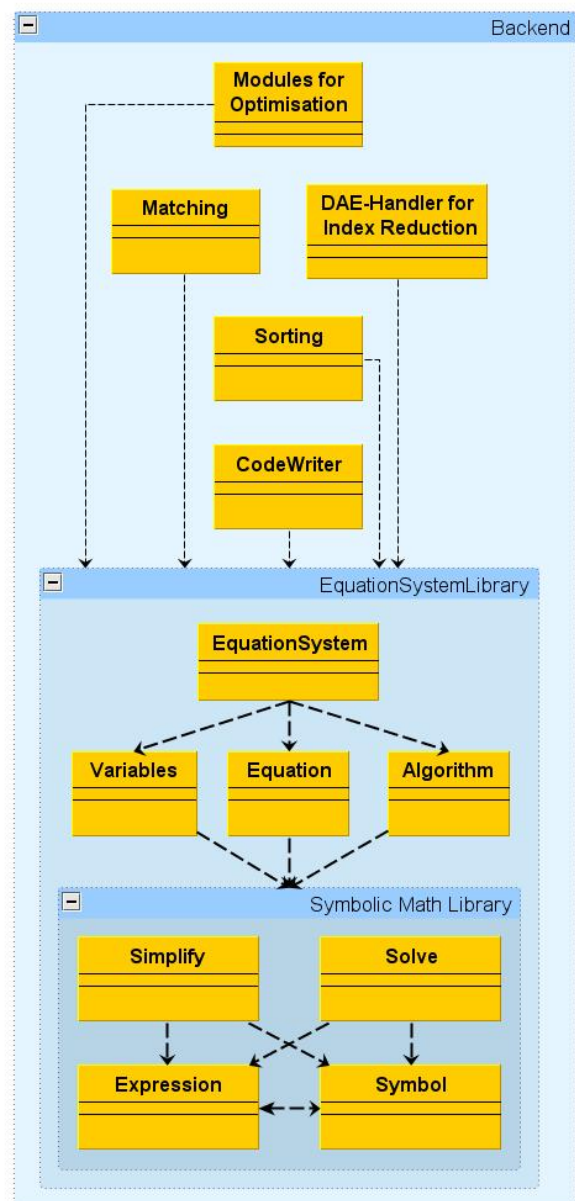


Figure 4. Concept for Implementation.

The developer can use the library functions of the equation system library and deal with the tasks of the optimisation algorithm.

The same principle holds for the box “DAE-Handler for Index Reduction” shown in Figure 4. For example to implement a classical handler for index reduction the developer can use the function to get the derivatives of the constrained equations and the function to replace equations. The classical handler for index reduction performs index reduction by taking the derivatives of the constrained equations.

The two modules to perform the sorting of the equations in BLT-Form and the module for code writing use the functions of the Symbolic Math Library and the equation system library as well as all other modules within the third level of the backend.

Furthermore, through the use of the library functions the implementation of debugging functionality for the system of equations is simplified and less error-prone because the information about the performed transformations would be added automatically by the library functions.

7 Interfaces and User Modules

With the possibility to export the system of equations during each phase the usability of the compiler for various analysis tasks can be significantly increased.

The next level of usability can be reached with an import of equation systems. This opens up the possibilities to:

- use the backend without the frontend,
- use other optimisation tools via an intermediate file format or an interface and
- store equation systems in different optimisation stages and run the optimisation algorithm again with improved optimisation modules.

As shown in Figure 5 the export is feasible by one or more modules in the pre- or past-optimisation phase. In Figure 5 an export to an XML-file format is shown as one example. The import has to be realized in the input phase to enable a consistency check of the system of equations.

Furthermore the presented design concept does not only include external interfaces as presented above. Additionally internal interfaces expand the usability of the compiler. With the advantages of the modularisation:

- replacement of modules,
- reusability of functions,
- distribution of the implementation complexity,
- limitation of complexity and

- a comfortable way to implement symbolic manipulation algorithms

the barriers to design, implement and test new algorithms for symbolic manipulation are reduced. Clearly, the symbolic simplification of a Modelica model is a difficult topic and it seems that only the Modelica compiler writers are able to provide such features. But, with a simplified access to the Modelica compiler development the Modelica compiler writer community can increase.

A comfortable way to implement those algorithms can be supported with the presented class concept from Figure 4, a development environment and the possibility to test the implemented algorithms. With the use of MetaModelica [8] for the Modelica compiler development it would be possible for an advanced Modelica user to implement his/her own modules. MetaModelica is an extension of the Modelica Language for Meta-programming facilities. Because of the similarity of MetaModelica and Modelica the additional requirements are marginal. Using the OpenModelica Development Environment (OMDev) the requirements mentioned above would also be fulfilled.

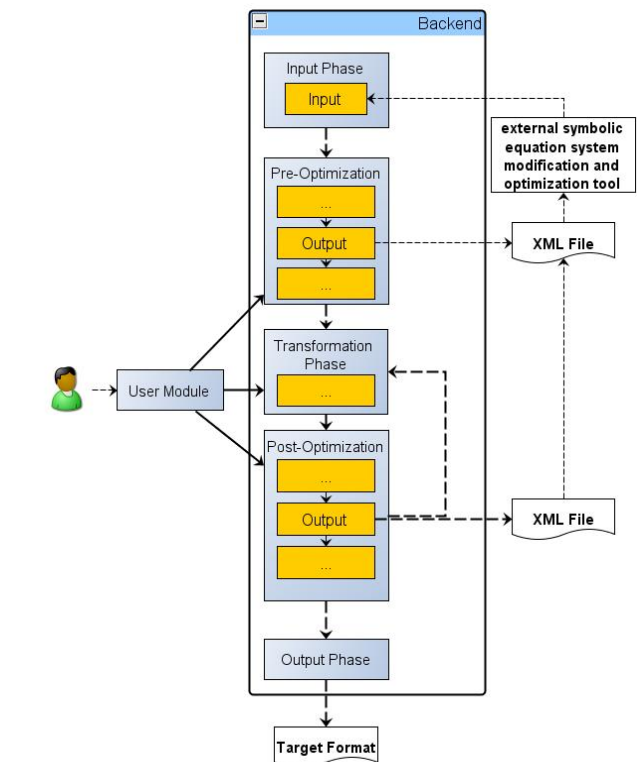


Figure 5: Interfaces for a modular Modelica compiler backend.

oment Environment (OMDev) the requirements mentioned above would also be fulfilled.

By offering the possibility to load the modules for symbolic manipulation dynamically during compile time the user could combine the compiler backend with

his/her own modules. With a standardized data object format it would furthermore be possible to implement new modules in a different language, e.g. C/C++.

8 Conclusion

With the possibility for the user to define and use his/her own symbolic manipulation algorithms the possibilities of using Modelica for many different kinds of applications beside simulation can be significantly increased.

Users may on their own develop, analyse and get a deeper understanding of symbolic manipulation algorithms and get better results for specific applications.

Furthermore, the user could e.g. get access to higher index equation system solvers without the challenge to implement his/her own compiler.

By giving the user a safe and easy option to extend and adapt the functionality of the compiler to their own needs, it would be possible to enhance the development of the compiler to a new quality and wider practicability. This paper presents a contribution to an approach towards achieving this goal.

The described concept is partly implemented in OpenModelica 1.6. The modularization has been achieved, but the external API with the XML interface is still mostly future work, and an internal API for flexible phase ordering is also future work.

References

- [1] Peter Fritzson: Principles of Object-Oriented Modelling and Simulation with Modelica 2.1, Page 57ff, Wiley IEEE Press, 2004.
- [2] Roberto Parrotto, Johan Åkesson, and Francesco Casella: An XML representation of DAE systems obtained from continuous-time Modelica models, EOOLT 2010.
- [3] 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
- [4] S.E. Mattsson, H. Olsson and H. Elmqvist: Dynamic Selection of States in Dymola. Modelica Workshop 2000 Proceedings, pp. 61-67,
- [5] P. Kunkel and V. Mehrmann: Index reduction for differential-algebraic equations by minimal extension, ZAMM, vol. 84, pp. 579–597, 2004.
- [6] F. E. Cellier, E. Kofman: Continuous System Simulation, Springer, 2006.
- [7] Bonvini, Donida, Leva: Modelica as a design tool for hardware-in-the-loop simulation, Proceedings 7th Modelica Conference, Como, Italy, Sep. 20-22, 2009
- [8] A. Pop and P. Fritzson: 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.
- [9] P. Fritzson, A. Pop, D. Broman, P. Aronsson: Formal Semantics Based Translator Generation and Tool Development in Practice. In *Proceedings of 20th Australian Software Engineering Conference (ASWEC 2009)*, Gold Coast, Queensland, Australia, April 14 – 17, 2009.
- [10] R.E. Tarjan: Depth First Search and Linear Graph Algorithms, SIAM Journal of Computing, 1, pp. 146-160, 1972.
- [11] H. Lundvall and P. Fritzson: Event Handling in the OpenModelica Compiler and Run-time System. In Proceedings of the 46th Conference on Simulation and Modelling of the Scandinavian Simulation Society (SIMS2005), Trondheim, Norway, October 13-14, 2005. An extended version in Linköping University Press, www.ep.liu.se.

Media Modeling for Distillation Process with Modelica/MWorks

Chen Chang Ding Jianwan Chen Liping Wu Yizhong
 CAD Center, Huazhong University of Science and Technology, Wuhan, China, 430074
 chenchang1210@hotmail.com jwdingwh@gmail.com chenlp@hustcad.com
 cad.wyz@hust.edu.cn

Abstract

This paper describes a package which consists of media models for distillation process with the method of classical two-parameter cubic thermodynamic equation of state in Modelica on platform MWorks. A structuralized and hierarchical modeling strategy was proposed in order to separate the components and the mixing rules of mixture from the vapor-liquid equilibrium predicting methods. With this strategy, a new media model can be introduced by changing the components' information of mixture and of mixing rule. An air distillation column model was built to test the air media model consists of nitrogen, oxygen and argon. The results obtained with MWorks were compared with simulation calculated with Aspen Plus. A very good agreement was found and the air model for distillation in the column worked well.

Keywords: media model; distillation; Modelica; Vapor-Liquid equilibrium

1 Introduction

Distillation process utilizes the different boiling point of components of mixture on the same condition to separate components from mixture. It is widely used in petroleum refining, chemical, metallurgical industries and so on. Up to the present there are numerous commercialized platforms like Aspen Plus and HYSYS that could be used to predict distillation process. These platforms mainly contain media model for vapor-liquid equilibrium (VLE), physicochemical constants database, process unit models, User Interface and algorithms library. Therein the media model is one of the most important cores of these tools. Its computational efficiency and accuracy have a strong impact on the simulation consumed time and accuracy of the whole process system model. The prediction methods of vapor-Liquid equilibrium in these tools are mainly fugacity coefficient method and fugacity- activity

coefficient method. With the former method the fugacity coefficients of vapor phase and liquid phase are calculated from the thermodynamic equation of state. The fugacity coefficient of vapor is from the thermodynamic equation of state (EoS) and the activity coefficient from excess function with the later method. The users of these tools just know how to use vapor-liquid equilibrium model but don't know the mechanism of modeling in these tools.

At present, there are lots of media models for pure substance and mixture in the Modelica Standard Library (MSL) *Media*[1]. But there is no media model for vapor-liquid equilibrium. In this paper, a package named *MediaForSapera* which contains physical property computing with the two-parameter EoS method for vapor-liquid equilibrium was built based on Modelica in MWorks[2]. The models in this package are based on general concept which allows the introduction of new working media for distillation by modification of a new model parameters and addition of particular equations of different EoS methods. The media package consists of parts as follows:

- 1、 Interfaces: consists of the partial media models that contain the common variables, constants, parameters and model structure. At present there is a base partial model and four partial models with two-parameter cubic EOS methods in the package.
- 2、 FluidData: consists of records contain the basic physicochemical constants, like molecular weight and critical temperature, and experimental regression coefficients of pure component of the mixture, such as coefficients of polynomial for specific enthalpy calculation.
- 3、 Examples: some mixture models for Vapor-Liquid Equilibrium, such as air model which is made up of nitrogen, oxygen and argon, simple air model composed of nitrogen and oxygen. All examples in the package are extends from package Interfaces.

- 4、 Tests: some simple models that are used to test the media models in Examples.

2 Methods of VLE based on the two-parameter EoS

Vapor-liquid equilibrium is a condition where a liquid and its vapor are in equilibrium with each other, a condition or state where the rate of evaporation equals the rate of condensation on a molecular level. It could be described by fugacity's equality of two phases as follow:

$$f_i^V = f_i^L \quad (1)$$

Where f_i^V and f_i^L means the fugacity in vapor and liquid phase of component i respectively. There are two methods that are used to calculate the fugacity. One is the fugacity-activity coefficient method. Another is fugacity coefficient method which predicts the fugacity of two phases by EoS mean as follows:

$$\begin{cases} f_i^V = \Phi_i^V y_i p \\ f_i^L = \Phi_i^L x_i p \end{cases} \quad (2)$$

Where Φ_i^V and Φ_i^L means the fugacity coefficient in vapor and liquid phase of component i respectively and y_i and x_i the mole fraction in vapor and liquid phase of component i respectively.

The fugacity coefficient method has good thermodynamic consistency and doesn't need the fugacity of standard state when it is calculated. Its results agree with the experimental measurements very well for a range of mixture except the strong polar molecule and electrolyte system. So it is widely applied to the prediction of vapor-liquid equilibrium of mixture. The two-parameter cubic EoS method is introduced into this paper to model VLE of mixture.

2.1 Two-parameter EoS

The thermodynamic equation of state has been proposed which has practical utility for the first time by Van der Waals in year 1873 and it has been developed rapidly in the next one hundred years. The EoS has been widely used in VLE calculation in large range of mixture, even includes strong polar molecule, during the latest decades. Herein the two-parameter EoS, such as Soave-Redlich-Kwong (SRK) equation, Redlich-Kwong equation (RK), Peng-Robinson (PR) equation and Harmens equation, has characteristic of a small number of parameter, simple expressions, easily-solving and exact results. So it has been developed very well and applied more

widely than other EoS.

In this paper, four kinds of two-parameter cubic EoS were introduced to model the VLE of mixture. They are RK, SRK, PR and Harmens and they could be expressed in a unified form:

$$p = \frac{RT}{v-b} - \frac{a(T)}{v^2+mbv+nb^2} \quad (3)$$

Where, p is the pressure of the media, pa . T the temperature of the media, K . v molar volume, m^3/mol . R the universal gas constant, $8.314J/(mol \cdot K)$. $a(T)$ and b the parameters of EoS, these two parameters of mixture are computed by mixing rule. m, n different constants according the different EoS. The fugacity coefficient and specific enthalpy of mixture are the important medium properties for distillation process simulation. For two-parameter cubic EoS method, they are computed as follows:

$$RT \ln \phi_i = \int_V^\infty \left[\left(\frac{\partial p}{\partial n_i} \right)_{T,V,n_j} - \frac{RT}{v} \right] dv - RT \ln z \quad (4)$$

$$h_i = \int_V^\infty \left[p - T \left(\frac{\partial p}{\partial T} \right)_{V,n_T} \right] dV + pV + h_i^0 \quad (5)$$

$$h = \sum y_i h_i \quad (6)$$

where, z means the compressibility factor, $=pv/(RT)$. h_i the actual specific enthalpy of component i of mixture. h_i^0 the ideal specific enthalpy of component i of mixture. h the actual specific enthalpy of mixture.

2.2 mixing rule

When the equation (3) is applied to mixture, the parameters $a(T)$ and b should be computed by some mixing rules that could be found in reference[3]. For $a(T)$ and b in RK, SRK and PR EoS, the Reid mixing rule is used usually.

$$\begin{cases} a_m = \sum_i \sum_j y_i y_j \sqrt{a_i a_j} (1 - k_{ij}) \\ b_m = \sum_i y_i b_i \end{cases} \quad (7)$$

Where, a_m and b_m mean the parameters of EoS for mixture. a_i , b_i and a_j the parameters of components i and j . y_i and y_j the mole fraction of components i and j in vapor or liquid phase of the mixture. k_{ij} is binary interaction coefficient between component i and component j . It can be obtained from experiment. The mixing rule for Harmens EoS is different from the other three EoS. It can be consulted in reference[4].

3 Implement in MWorks

There are large numbers of mixture in the distillation process industries and so are the media properties

prediction methods and the mixing rules of introduction of new working media for distillation by components in mixture. In order to allow the modifying the minimizing set of parameters and

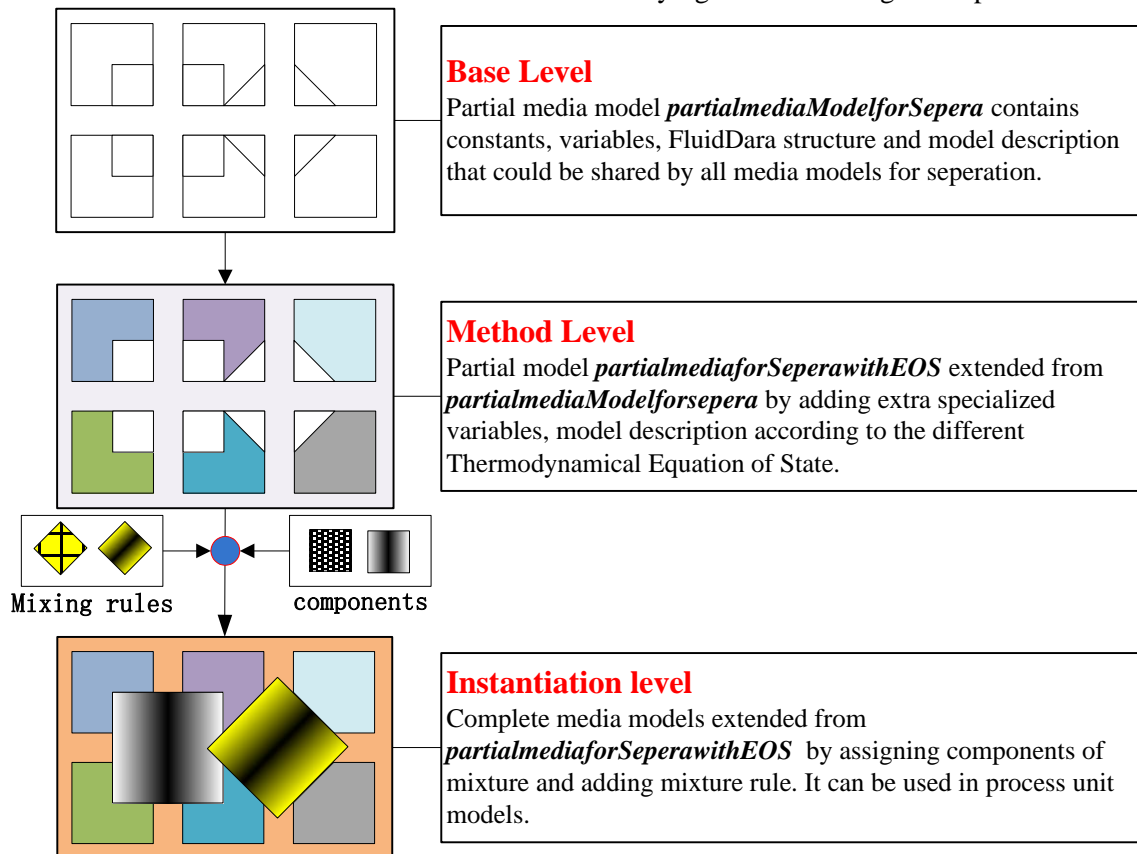


Figure 1 Sketch of modeling strategy of media model for distillation

adding the minimizing set of equations, a structuralized and hierarchical modeling strategy was used for the modeling of media model package for distillation, as shows in figure 1. The package has three levels: base level, method level and instantiation level.

The base level model is a partial model named *PartialMediaForSepera* which all the media models for distillation are extended from. It consists of some common parts that can be shared by all models with the method of two-parameter cubic EoS. The more details are listed as follows.

- Medium constants: media name, names and number of chemical substances, critical properties, etc.
- FluidData structure: definition of record type **FluidConstants** used for storing physicochemical constants of pure substances as show in figure 2. All components' physicochemical constants used in VLE mixture have to comply with this data structure. When a new substance is introduced, the constants are assigned in order of the constants' sequence in figure 2.
- ThermodynamicState: a minimum set of

variables of state. It is used for function calling.

- BasePropertiesRecord: common variables could be shared by all media model with all EOS, such as pressure, specific enthalpy of vapor and liquid phase, compressibility factor, etc.
- BaseProperties: a partial model consists of common behavior of all media model with all EOS.
- Type definition: such as "Type CompressibilityFactor = Real(final quantity= "compressibility", min=0, max=1)".

```

record FluidConstants
  extends Modelica.Icons.Record;
  String name "Name of ideal gas";
  SI.MolarMass molarMass;
  SI.Temperature criticalTemperature;
  SI.AbsolutePressure criticalPressure;
  SI.MolarVolume criticalMolarVolume;
  Real acentricFactor "acentric factor";
  CompressibilityFactor criticalcompressibility;
  Real[5] bh
    "coefficient of idea media SpecificEnthalpy";
end FluidConstants;

```

Figure 2 Structure of the FluidConstants

The method level model is extended from the base level model *PartialMediaForSepera* by adding some special variables and the description according to the different two-parameter cubic EoS methods. For instance, the PR EoS method has the expression of the fugacity coefficient in vapor phase as follow.

$$\ln\phi_i = \frac{b_i}{b_m} (z - 1) - \ln(z - B_m) - \frac{A_m}{2\sqrt{2}B_m} \left[\frac{2\sum_j x_j a_{ij}}{a_m} - \frac{b_i}{b_m} \right] \ln \left(\frac{z + (1 + \sqrt{2})B_m}{z + (1 - \sqrt{2})B_m} \right) \quad (8)$$

Equation (8) is a special equation which is only suit for the PR EoS but not other. And it should be added to the BaseProperties section when the partial method level media model with the PR method is built by extending the partial base level model, as shows in figure 3.

Finally the case level models are constructed by extending the method level models and adding the components and the mixing rule of mixture. For example, the air model for distillation was built by defining the medium name, components' names and appointing the fluid data of Nitrogen, Oxygen and Argon and the binary interaction coefficient of components as figure 4.

As discussed above, the reason for introducing the structuralized and hierarchical modeling strategy is that we want to separate the components information of mixture and the mixing rules from VLE predicting method. The same VLE predicting method can be applied to the different mixtures by changing the components information of mixture and mixing rule like the air model showed in figure 4. A user just needs to know the physicochemical constants of components and the mixing rule when the user wants to introduce a new model.

```

redeclare replaceable partial model extends
BaseProperties
  ...
equation
  ...
  Log(FugacityCoefficientV) = b .* (zV - 1) /
  bV_Mix .- Modelica.Math.log((zV - BV_Mix)) -
  AV_Mix / BV_Mix / (2 * sqrt(2)) * (exmav /
  aV_Mix .- b / bV_Mix) * (Modelica.Math.log(((zV +
  2.414 * BV_Mix) / (zV - 0.414 * BV_Mix))));
  ...
end BaseProperties;

```

Figure 3 The media model with PR EoS method by extending the base model

```

package N2O2ArWithPRMethod
  extends
  Interfaces.PartialMediaForSeperaWithPRMethod(
    mediumName="Air for seperation",
    substanceNames={"O2", "Ar", "N2"},
    ...);
  redeclare model extends BaseProperties
  equation
  PureFluidConstants={FluidData.O2,FluidData.Ar,
  FluidData.N2};
  kij={{0,0.01396,-0.01238},
        {0.01396,0,-4.071e-3},
        {-0.01238,-4.071e-3,0}};
  end BaseProperties;
end N2O2ArWithPRMethod;

```

Figure 4 Air model with PR EoS method

4 Test case

In order to validate the media model, a steady state air distillation column model was built in Modelica on MWorks based on the assumptions as follows.

- Every column theoretic stage is considered as an adiabatic system.
- In the energy balance the wall material's heat storage is neglected.
- Chemical reactions between Nitrogen, Oxygen and Argon are not considered.

The column contains eighteen theoretic stages and the first of them is the condenser which is at the top of column. The cooled feed air flows into the fifteenth stage. Each theoretic stage obeys the mass and energy balance.

$$\sum \dot{m}_i = 0 \quad (9)$$

$$\sum \dot{m}_i h_i = 0 \quad (10)$$

Where, \dot{m}_i means the mole flow rate on the inlet and outlet of the theoretic stage. h_i the specific enthalpy correspond to the \dot{m}_i .

The media model used in the column model was the model *N2O2ArWithPRMethod* mentioned in the

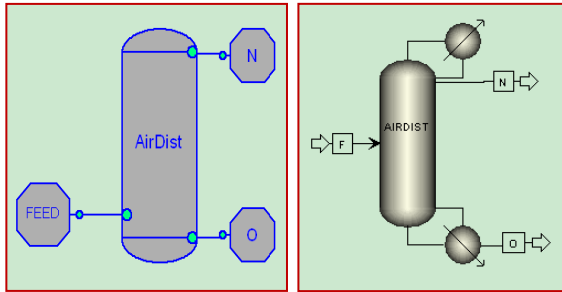


Figure 5 Air distillation column in MWorks (left) and Aspen Plus (right)

third section. The results of this model were validated with calculations simulated with tool Aspen Plus. The figure 5 shows the sketch of air distillation column model in MWorks and Aspen Plus. **F** means the cooled feed. **N** means nitrogen product. **O** means oxygen-enriched liquid air. For the case the following input values and parameters are used.

Table 1 input values and parameters in the model

Feed pressure	6[bar]
Feed total flow	200[kmol/hour]
Feed mole fraction	0.785[N ₂], 0.205[O ₂],

	0.01[Ar]
Feed stage	No.15
Stage numbers	18
Condensation rate	100%
Condenser pressure	5[bar]
stage pressure drop	0.01[bar]
Distillate rate	50[kmol/hour]
Reflux	1.3

The model of air separation tower was simulated with 18 theoretical stages and the input values and parameters used in MWorks and Aspen Plus are listed in table 1. The physical property model in Aspen Plus is also Peng-Robinson method.

Figure 6 shows the temperature of medium in each stage from the simulation in Aspen Plus and MWorks. The 18th theoretical stage is the oxygen-rich tray and the first stage is the condenser. The maximum difference value of the temperature is 0.201K and the mean value of that is 0.1482K between the results from MWorks and that from Aspen Plus. Figure 7 demonstrates the values of the mole fraction of components (nitrogen, oxygen and argon) of air in gas phase in each stage from the simulation in MWorks and in Aspen Plus. The maximum difference of mole fraction of nitrogen is 0.1826% between the results from MWorks and that from Aspen Plus. The mole fraction of oxygen 0.1820%. The mole fraction of argon 0.0049%. The average difference of mole fraction of nitrogen is 0.0778%

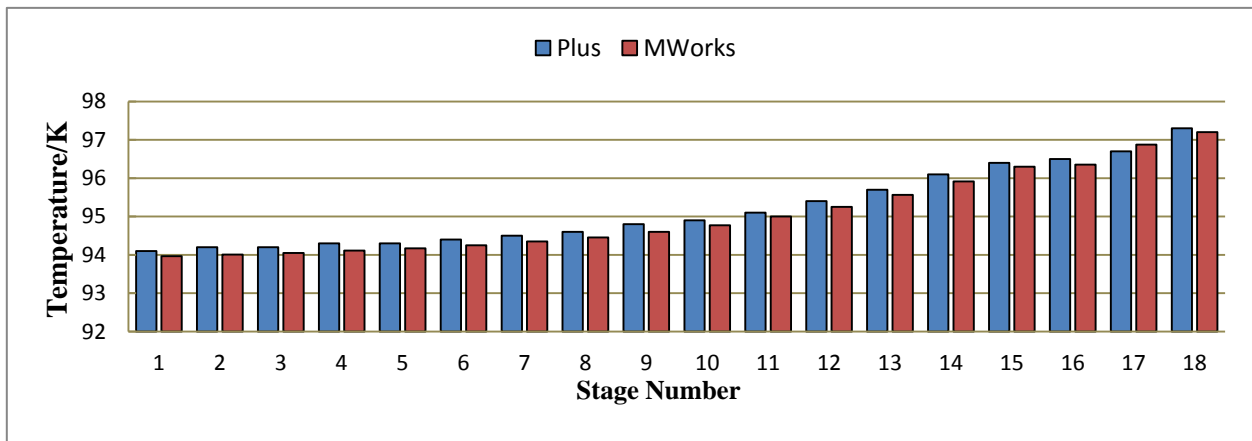


Figure 6 Temperature in each stage

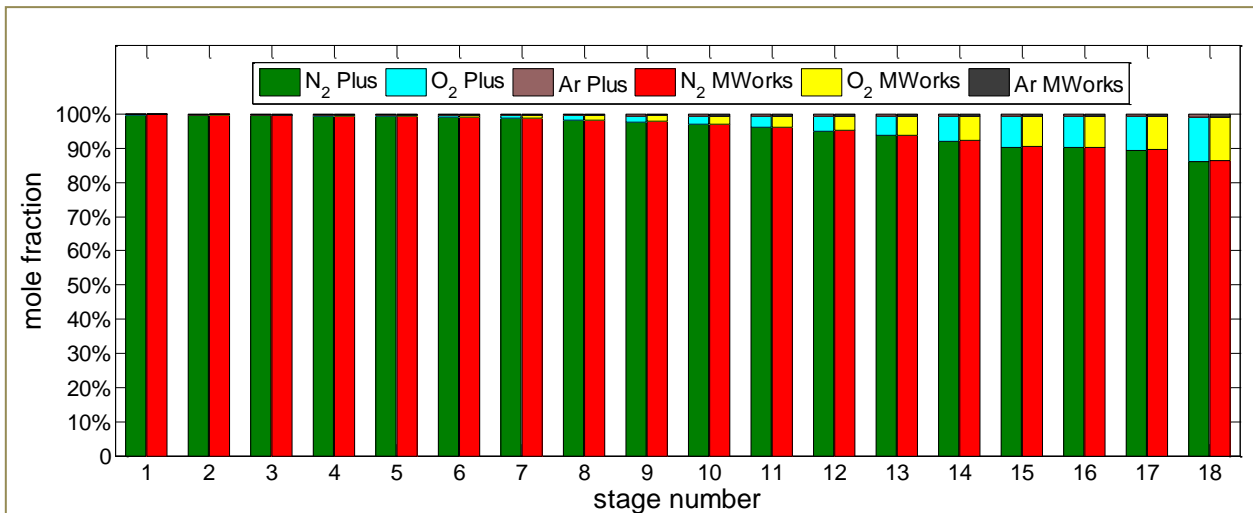


Figure 7 Vapor phase mole fraction profile of air distillation column

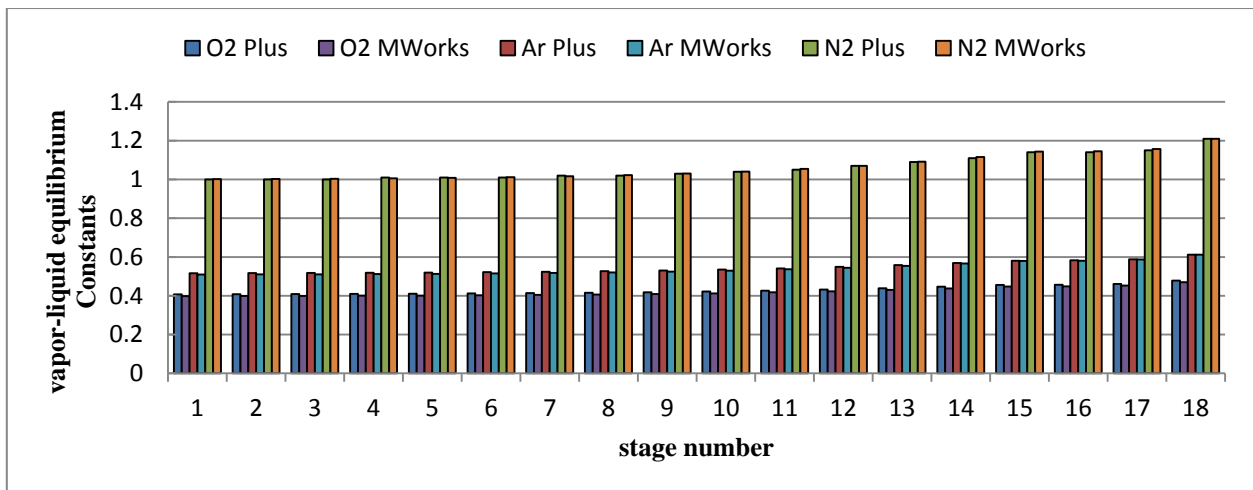


Figure 8 vapor-liquid equilibrium constant profile of air distillation column

between MWorks and Aspen Plus. The mole fraction of oxygen 0.0672%. The mole fraction of argon 0.0025%.

The vapor-Liquid equilibrium constants of substances of mixture simulated in each stage in Aspen Plus and MWorks are showed in Figure 8. The maximum relative difference of the equilibrium constant of nitrogen is 0.6078% between the results from MWorks and that from Aspen Plus. The equilibrium constant of oxygen 2.4411%. The equilibrium constant of argon 1.4253%. The mean relative difference of the equilibrium constant of nitrogen is 0.2556% between MWorks and Aspen Plus. The equilibrium constant of oxygen 2.1032%. The equilibrium constant of argon 0.8879%.

Summing up the above, the results obtained in the MWorks model show similar results compared to the Aspen Plus and the media model of air for distillation based on Modelica in MWorks works well.

5 Conclusions

Distillation process widely exists in Petroleum refining and chemical industry, et al. The model for vapor-liquid equilibrium predicting is one of the important parts in distillation process. Its computational efficiency and accuracy have a strong impact on the computational efficiency and accuracy of process model. In this paper, a media model package, based on the structuralized and hierarchical modeling strategy, was built for distillation process in order to explore a suitable modeling strategy for media model. An air model with this strategy was used in the air separation column model and the results were validated by the simulation in Aspen Plus. The differences of interested variables' value between the simulation of MWorks and of Aspen Plus are slight and the air model works well.

At present, the method for media model is

two-parameter cubic EoS and the number of substances is small. The future work is adding the other EoS and activity coefficient method for media model and also substances to make the media package more generally applicable.

Acknowledgments

The paper was supported by National Nature Science Foundation of China (No. 60704019, No.60874064), Major State Basic Research Development Program of China (No. 2011CB706500).

References

- [1] Modelica Association. Modelica Standard Library, February, 2008. Version 3.0.
- [2] Zhou F, Zhang H, Zhu H, et al. Design and Implementation of Animation Post-processor Based on ACIS and HOOPS in MWorks. Proceedings of the 7th International Modelica Conference, Como, Italy, September 20-22, 2009, pp. 663-668.
- [3] Reid R C, Prausnitz J M, Poling B E. The properties of gases and liquids. 4th ed. New York: McGraw-Hill, 1987.
- [4] Harmens. A Cubic Equation of State for the Prediction of O₂-Ar-N₂ Phase Equilibrium, Cryogenics, v. 17, pp. 519-521, 1977.

Experimental comparison of the dynamic evaporator response using homogeneous and slip flow modeling

Martin Ryhl Kærn^{†,‡} Brian Elmegaard[†] Lars Finn Sloth Larsen[‡]

[†]Technical University of Denmark, Department of Mechanical Engineering
Nils Koppels Allé Bygn. 403, DK-2800 Lyngby, Denmark, e-mail: pmak@mek.dtu.dk

[‡]Danfoss A/S, Refrigeration and Air-Conditioning
Nordborgvej 81, DK-6430 Nordborg, Denmark, e-mail: martin@danfoss.com

Abstract

The dynamic response from an evaporator is important for control of refrigeration and air-conditioning systems. Essentially, the prediction of refrigerant charge inside the evaporator is crucial for the dynamic behavior. The prediction of refrigerant charge follows from suitable void fraction correlations from the literature. A chosen set of void fraction correlations (slip flow) and the assumption of homogeneous flow will be investigated in this paper and compared to experiments on a simple coaxial type evaporator. The numerical model of the evaporator is a dynamic distributed mixture model, where different void fraction correlations can be applied. It is shown that the dynamic response of the homogeneous model is too fast, whereas the slip flow models agree well with the experiments. Another difference is that the charge prediction of the homogeneous model is approximately 2-3 times less than the slip flow models.

Keywords: refrigeration; air-conditioning; evaporator; two-phase flow; modeling; Modelica; transient; dynamic; simulation

Nomenclature

Roman

A	Cross-sectional area (m ²)
c_p	Specific heat capacity (J kg ⁻¹ K ⁻¹)
D	Inner tube outer diameter (m)
d	Inner tube inner diameter (m)
F_w	Wall friction force (N m ⁻³)
G	Mass flux (kg m ⁻² s ⁻¹)
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	Thermal conductivity (W m ⁻¹ K ⁻¹)
M	Mass (kg)
\dot{m}	Mass flow rate (kg s ⁻¹)
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)
\dot{V}	Volume flow rate (m ³ /s)
x	Vapor quality (-)
z	Axial channel length (m)

Greek

α	Void fraction (-)
ρ	Density (kg m ⁻³)
$\bar{\rho}$	Mixture density (kg m ⁻³)
ρ'	Momentum density (kg m ⁻³)
σ	Surface tension (N m ⁻¹)
θ	Angle to horizontal plane (deg.)

Subscripts

ax	Axial
b	Brine
f	Saturated liquid
g	Saturated gas
H	Homogeneous
rad	Radial
sat	Saturation
w	Wall

1 Introduction

Refrigerant charge minimization in refrigeration and air-conditioning systems is becoming increasingly important for environmental and legislative reasons. As the charge is minimized the dynamic behavior of the system becomes quicker and the requirements for the control increases. Furthermore, new control methods continuously evolve that requires more and more accurate prediction of the dynamic behavior of the evaporator. Thus there is a demand for accurate modeling of the evaporator response.

The focus in this paper is the modeling of the dynamic behavior in a dry-expansion evaporator with experimental validation. The key variable here is the void fraction. The void fraction essentially determines the refrigerant charge and thus the dynamic response of the evaporator. As we shall see in section 4, the use of a void fraction correlation determines the slip ratio of the two phases, implying a slip flow model. If the slip ratio is assumed to be unity, then the flow is homogeneous (i.e. both phases travel with the same velocity), implying the homogeneous model.

Many void fraction correlations exist in the literature. Some are rather simple analytical relations, others are quite sophisticated and of empirical nature. The question in mind is: When is it sufficient to use the homogeneous model, in contrast to the slip flow models?

Despite the large amount of work that has been done on the development of void fraction correlations, some void fraction correlations do not satisfy a smooth transition in void fraction at the two-phase to vapor phase transition. Woldeesemayat and Ghajar [1] compared 68 void fraction correlations in order to find an acceptable void fraction correlation that could predict most of the collected experimental data for all inclination angles, fluids and flow patterns. They developed a modified version of the Dix [2] model, however, the model does not ensure a smooth transition at the two-phase to vapor phase transition. For dynamic simulation the transition and its derivatives should be continuous or at least a smoothening may be used. Furthermore, the correlation complexity should be sought to a minimum while capturing the main dynamics of interest.

Woldeesemayat and Ghajar [1] also gave void fraction correlation recommendations, considering each specific type of flow. For horizontal flow, as is the case of consideration in this paper, the void fraction of Premoli et al. [3] was worth the general recommendation among others, regardless of flow regime and fluids. Recently, the same void fraction model was recommended by Maa et al. [4] and Mader et al. [5] as the

best choice for R410A air-conditioners. The model also ensures a smooth transition at the two-phase to vapor-phase transition, and for these reasons the Premoli et al. [3] model will be used in this study.

Wojtan et al. [6] recommends Steiners version of the Rouhani and Axelsson void fraction model [7], and uses this model in their general flow map [8] for predicting both two-phase heat transfer and pressure drop in a flow regime dependant way. The earlier versions of the Rouhani and Axelsson model did not ensure a smooth transition from two-phase to vapor phase, however Steiner modified the model for horizontal flow in a way that ensures a smooth transition. The Steiner version of the Rouhani and Axelsson model is also included in this study.

The two aforementioned void fraction correlations are both sophisticated models and functions of pressure, p , vapor quality, x , mixture mass flux, G , surface tension, σ , and acceleration of gravity, g . Simpler models only dependant on pressure and vapor quality exists, e.g. the Zivi [9] model, which is one of the simplest void fraction models. The inclusion of the Zivi [9] model fulfills our set of slip flow models to be investigated in this paper.

The paper includes a brief description of the experiments, the test rig and the numerical modeling framework. Then the results of the transient evaporator response are addressed.

2 The experiments

The experimental data are obtained from the master thesis of Antonius [10], who compared the experimental results with commercial software Sinda/Fluint [11]. The thesis is written in Danish, however the main results are given in English in Jakobsen et al. [12]. Sinda/Fluint is a general thermo-fluid network analyzer capable of simulating static and dynamic behavior of multi-phase fluid networks as they interact with thermal structures, using a lumped parameter finite-difference approach. It is quite similar to the numerical model introduced in section 4, however the empirical correlations for heat transfer, friction coefficient and void fraction are restricted to a predefined possibilities.

The test case geometry and boundary conditions were kept as simple as possible in order to focus on the two-phase flow. Figure 1 shows a sketch of the test case coaxial evaporator and corresponding boundary conditions for the numerical simulations. The outer tube wall is insulated to minimize heat flow from the

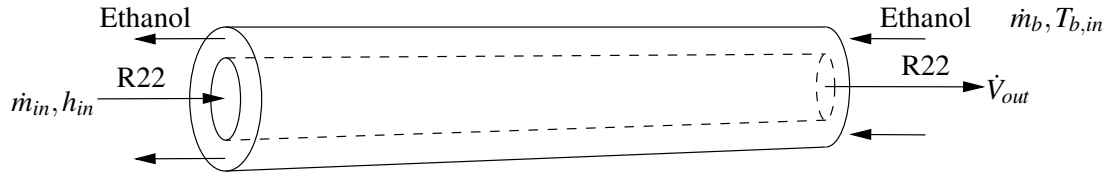


Figure 1: Sketch of test case evaporator

surroundings. R22 is the refrigerant flowing in the inner tube, whereas ethanol with 10% water by mass is flowing in countercurrent direction in the outer tube shell.

Both a step in refrigerant mass inflow and volume outflow was considered in the work by Antonius [10]. The original data does unfortunately no longer exist, however, the boundary conditions as function of time, as indicated on figure 1, was curve fitted in the work to be used as input to Sinda/Fluint. Only one set of these curve fits is available in the thesis, whereas the rest are missing. For these reasons it is only possible to reproduce the evaporator response on a step change in volume outflow. The dynamics of the mass outflow and outlet pressure is then compared to the homogeneous model and slip flow models. The mass inflow, mass outflow and outlet pressure are obtained from figures in Antonius [10] by graphical means for the comparison.

3 The test rig

A schematic of the test rig is shown in figure 2. The test case coaxial evaporator is 7 m long and made of copper. The inner tube has an internal and outer diameter of 11.1 and 12.7 mm, respectively, and the outer tube outer diameter is 20.18 mm.

The following data are used to obtain the appropriate boundary conditions as shown in figure 1. At the inlet of the evaporator, the mass flow is measured directly by M2, the mixed-cup enthalpy h is found using the subcooled liquid temperature T3 and pressure P1. The volume flow at the outlet of the evaporator is found using the mass flow M1 and the density of the superheated vapor at temperature T1 and pressure P3. The mass flow M3 and temperature T4 are directly applicable as boundary conditions for the brine system.

A thorough documentation of the test rig (e.g. apparatus, calibration and data acquisition method) can be found in Antonius [10].

4 Model formulation

The model is implemented in Dymola 7.4 [13]. Dymola is based on the Modelica language and 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 [14, 15]. Thermophysical properties for R22 are obtained from the Refeqns package [16]. Thermophysical properties for ethanol with 10% water by mass are obtained from VDI Wärmeatlas [17].

4.1 Refrigerant flow

The simplest form of the one-dimensional two-phase flow models is chosen, i.e. the mixture model as derived by performing a differential analysis on each phase and adding the phasic equations [18]. The result is the mixture mass conservation, the mixture momentum conservation and the mixture energy conservation equations given by

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

$$\frac{\partial \dot{m}}{\partial t} + \frac{\partial}{\partial z} \left(\frac{\dot{m}^2}{\rho' A} \right) = -A \frac{\partial p}{\partial z} - F_w A - \bar{\rho} g A \sin \theta \quad (2)$$

$$A \frac{\partial}{\partial t} (\bar{\rho} \bar{h} - p) + \frac{\partial}{\partial z} (\dot{m} h) = P q_w'' \quad (3)$$

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

$$\bar{\rho} = \rho_g \alpha + \rho_f (1 - \alpha) \quad (4)$$

$$\bar{h} = [\rho_f h_f (1 - \alpha) + \rho_g h_g \alpha] / \bar{\rho} \quad (5)$$

$$h = (1 - x) h_f + x h_g \quad (6)$$

$$\rho' = \left(\frac{(1 - x)^2}{\rho_f (1 - \alpha)} + \frac{x^2}{\rho_g \alpha} \right)^{-1} \quad (7)$$

where the void fraction is defined as $\alpha = A_g/A$, and the vapor quality is defined as $x = \dot{m}_g/\dot{m}$.

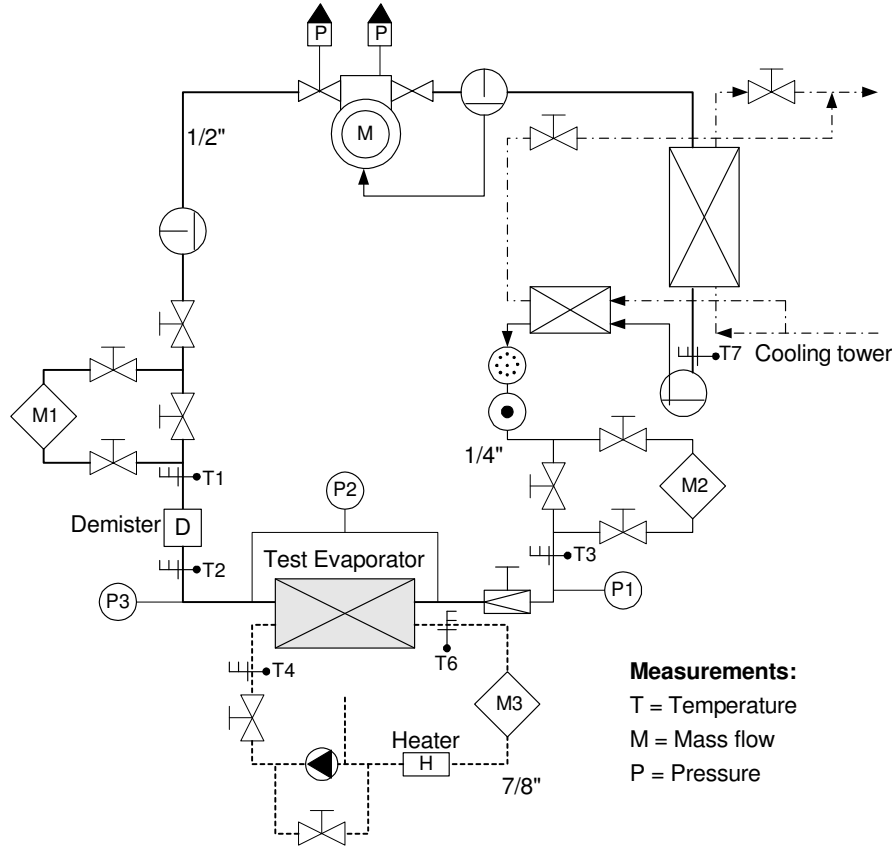


Figure 2: Evaporator test rig

Using the definition of the slip ratio, the void fraction and the vapor quality, the fundamental void-quality relation can be derived as

$$\begin{aligned}
 S &= \frac{U_g}{U_f} = \frac{\frac{\dot{m}_g}{\rho_g \alpha A}}{\frac{\dot{m}_f}{\rho_f (1-\alpha) A}} \\
 &= \frac{x}{1-x} \frac{\rho_f}{\rho_g} \frac{1-\alpha}{\alpha}
 \end{aligned} \quad (8)$$

and rewritten in terms of the void fraction as

$$\alpha = \left[1 + \frac{\rho_g}{\rho_f} \frac{1-x}{x} S \right]^{-1} \quad (9)$$

If homogeneous flow is assumed, then $S = 1$ and the homogeneous void fraction, α_H , may be calculated by equation 9. 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

$$\rho_H = \left(\frac{x}{\rho_g} + \frac{1-x}{\rho_f} \right)^{-1} \quad (10)$$

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

$$\frac{\partial \bar{\rho}}{\partial t} = \frac{\partial \bar{\rho}}{\partial p} \bigg|_{\bar{h}} \frac{\partial p}{\partial t} + \frac{\partial \bar{\rho}}{\partial \bar{h}} \bigg|_p \frac{\partial \bar{h}}{\partial t} \quad (11)$$

where the partial derivatives of mixture density with respect to pressure and in situ enthalpy are calculated by numerical finite difference as

$$\frac{\partial \bar{\rho}}{\partial p} \bigg|_{\bar{h}} = \frac{\bar{\rho}(p + \Delta p, \bar{h}) - \bar{\rho}(p, \bar{h})}{\Delta p} \quad (12)$$

$$\frac{\partial \bar{\rho}}{\partial \bar{h}} \bigg|_p = \frac{\bar{\rho}(p, \bar{h} + \Delta \bar{h}) - \bar{\rho}(p, \bar{h})}{\Delta \bar{h}} \quad (13)$$

Equations 1, 2 and 3 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 [19]. 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 3, where ψ denotes

a thermodynamic quantity and $\hat{\psi}$ its approximation. Similar discretization methodology was used in [20].

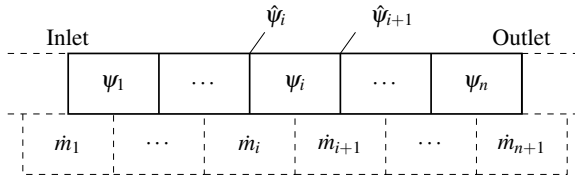


Figure 3: Staggered grid structure; thick = control volume grid, dashed = staggered grid

The mass and energy conservation equations become

$$A\Delta z \frac{d\bar{p}_i}{dt} = \dot{m}_i - \dot{m}_{i+1} \quad (14)$$

$$A\Delta z \frac{d}{dt} (\bar{\rho}_i \bar{h}_i - p_i) = \dot{H}_i - \dot{H}_{i+1} + \dot{Q}_i \quad (15)$$

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

$$\hat{\psi}_i \approx \delta_i \psi_i + (1 - \delta_i) \psi_{i-1} \quad i = 1..n+1 \quad (16)$$

where δ_i is the indicator function denoting the direction of the mass flow

$$\delta_i = \begin{cases} 0 & \dot{m} \geq 0 \\ 1 & \dot{m} < 0 \end{cases} \quad (17)$$

The momentum equation becomes

$$\Delta z \frac{d\dot{m}_i}{dt} = \Delta \dot{I}_i - A(p_i - p_{i-1}) - F_{w,i} A \Delta z - \hat{\rho}_i g A \Delta z \sin \theta \quad (18)$$

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

$$\Delta \dot{I}_i \approx \frac{(\dot{I}_{i-1} - \dot{I}_i) + (\dot{I}_i - \dot{I}_{i+1})}{2} = \frac{d\dot{I}_{i-1} + d\dot{I}_i}{2} \quad (19)$$

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 other boundary conditions than the ones indicated on figure 1, i.e. \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 (if slip flow), must be supplied to close the system of equations.

4.2 Tube wall

The tube wall is discretized according to the Resistance Capacitance Method [21]. 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

$$M c_p \frac{dT}{dt} = \dot{Q}_W + \dot{Q}_E + \dot{Q}_S + \dot{Q}_N \quad (20)$$

where $\dot{Q}_S = -P\Delta z q''_w$ from equation 3. The entering and leaving heat flows are depicted on figure 4.

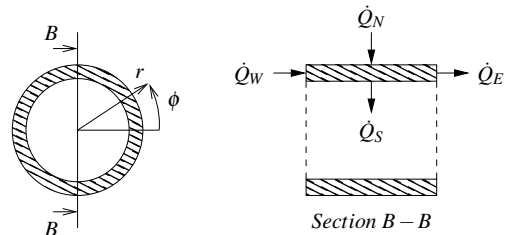


Figure 4: Heat flows to and from the tube wall

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

$$R_{ax} = 0.5 \frac{\Delta z}{kA} \quad (21)$$

$$R_{rad} = 0.5 \frac{\ln \frac{D/2}{d/2}}{2\pi k \Delta z} \quad (22)$$

The boundary condition at the inlet and outlet of the pipe wall is simply no heat flow in the axial direction.

4.3 Liquid flow

The liquid flow is assumed to be incompressible and cannot accumulate mass or energy. With these assumptions the mass and energy conservation equations for each liquid cell reads

$$\dot{m}_{in} - \dot{m}_{out} = 0 \quad (23)$$

$$(\dot{m}c_p T)_{in} - (\dot{m}c_p T)_{out} + \dot{Q}_N = 0 \quad (24)$$

Again Newton's law of cooling is applied to compute the heat transfer as

$$\dot{Q}_N = h_{tc} \Delta z P (T_w - T_{in}) \quad (25)$$

where the 1st order upwind approximation of the liquid cell temperature is used. A correlation for the heat transfer coefficient must be applied.

4.4 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 [15]. The used correlations are shown in table 1.

4.5 Heat exchanger architecture

Components of the refrigerant (both control volume grid cell and staggered grid cell), the wall and the liquid have been made in Dymola, and essentially arrays of these components are put together to form the evaporator in counter flow operation, as shown on figure 5.

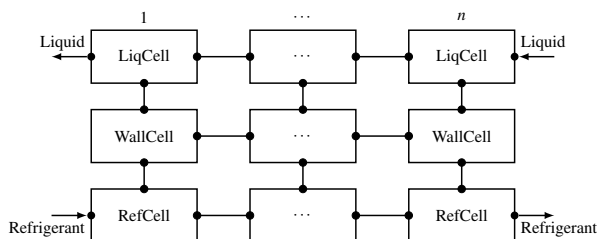


Figure 5: Heat exchanger architecture, counter flow.

We chose to use 30 cells in our simulations. Furthermore, we did not use any of the elements of the Modelica standard library. We chose this to learn every step of the implementation in Modelica and to be

Table 1: Overview of used correlations

Liquid brine	
Heat transfer	Dittus and Boelter [22]
Single phase refrigerant	
Heat transfer	Gnielinski [23]
Friction	Blasius [24]
Two-phase refrigerant	
Heat transfer	Shah [25]
Friction	Müller-Steinhagen and Heck [26]
Void fraction	Zivi [9] Premoli et al. [3] Steiners version of Rouhani and Axelsson [7]

able to quickly apply changes to the model formulation and correlations if necessary.

5 Results

In this section, the results are compared to the experimental data at a step decrease or increase in volume outflow. The cases correspond to a change in outflow by capacity control of the compressor. Firstly, we address the different void fraction correlations to be used.

5.1 Comparison of the void fraction models

All the used correlations for the void fraction (slip flow models) are shown in table 1.

Using equation 9 with $S = 1$ becomes the homogeneous model, where each phase travels with the same velocity. If we use the slip ratio correlation by Zivi [9] in equation 9, i.e. $S = (\rho_g/\rho_f)^{-1/3}$, we get the Zivi void fraction model. The Premoli et al. [3] model and the Steiners version of the Rouhani and Axelsson model [7] depend on both flow and fluid properties in more complicated ways. The void fraction models are compared in figure 6 as functions of vapor quality.

It is clearly seen that the area of the liquid is smaller in the homogeneous model, indicating a faster dynamic response compared to all the other models. The Premoli model and the Steiners version of the Rouhani and Axelsson model seem almost the same for the specific refrigerant and conditions. At low vapor qualities the Zivi model shows the largest amount of liquid, however at vapor qualities above 0.2, which is often the case for dry-expansion systems, it shows less liquid than the Premoli model and the Steiners version of the Rouhani and Axelsson model, indicating a faster

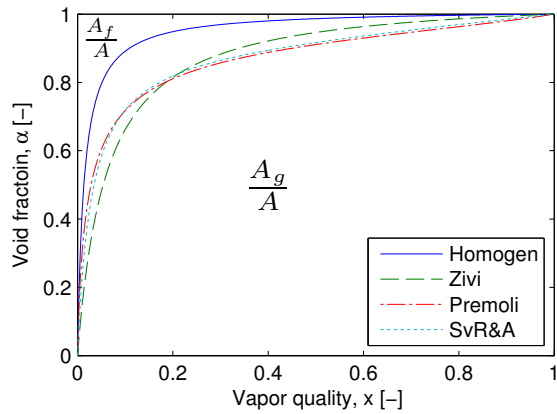


Figure 6: Comparison of the chosen void fraction models. (R22, $T_{\text{sat}} = -5.7^\circ\text{C}$, $p = 4.12$ bar, $G = 124$ $\text{kg m}^{-2}\text{s}^{-1}$)

dynamic response, however, not as fast as the homogeneous model.

5.2 A step decrease in volume outflow

In figure 7, the response of the mass outflow and the outlet pressure are compared at a step decrease in volume outflow.

At time equal 5 seconds the step occurs. The step shows an undershoot in mass outflow in the experiment and the models. The models have a sharp edge at the peak of the undershoot, which originates from the curve fitted boundary condition for volume outflow. After the undershoot the mass outflow increases and adjusts to a new steady state after approximately 40 seconds. Only the homogeneous model shows an overshoot during the transient response. Similarly, the pressure increases as the volume outflow decreases, however, only the homogeneous model shows an overshoot here.

5.3 A step increase in volume outflow

In figure 8, the response of the mass outflow and the outlet pressure are compared at a step increase in volume outflow.

At time equal 7 seconds the increase in volume outflow occurs. The step shows a quite large overshoot in the mass outflow, however, it adjusts quicker to the new quasi-steady state after approximately 25 seconds. Again the homogeneous model shows another undershoot after the overshoot. The pressure decreases on the step increase of volume outflow. Again the homogeneous model shows an undershoot in contrast to the other void fraction models.

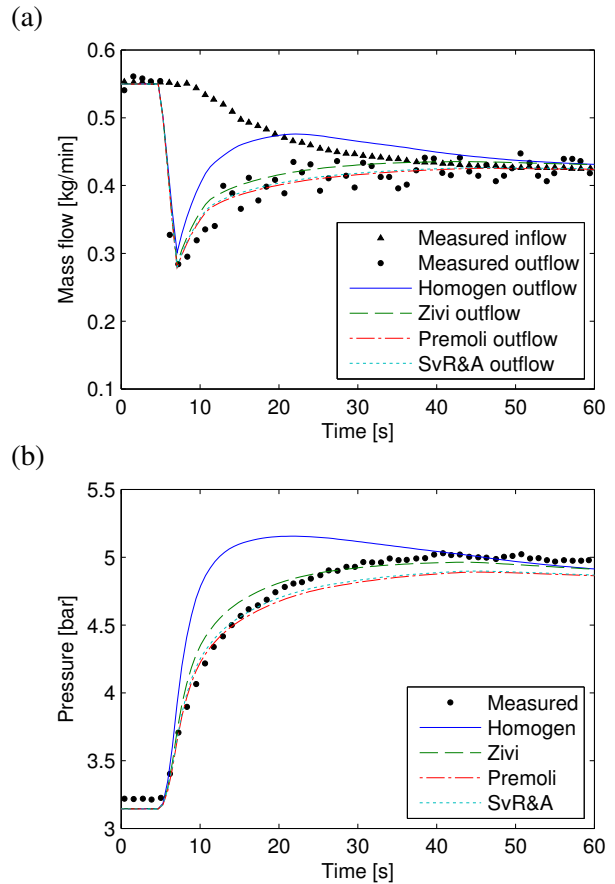


Figure 7: Mass outflow (a) and pressure outlet (b) response on decrease in volume outflow

5.4 Discussion

Apparently, the homogeneous model reacts too quickly. The response of the Premoli model and the Steiners version of the Rouhani and Axelsson model seem to predict almost the same and the most accurate responses from the evaporator. The Zivi model seem to be quite close to the experimental data, and can be considered as an easy way to capture the main dynamics of the evaporator, however, if the dynamics are more important, one should turn to the more sophisticated void fraction models.

Another observation is that the steady state values are not affected by the individual void fraction models. This is because the two-phase heat transfer and frictional pressure drop correlations are functions of vapor quality and not the choice of the void fraction correlation. Some more sophisticated two-phase heat transfer and frictional pressure drop correlations incorporate their own void fraction correlation in for example their heat transfer correlation [27]. One may inspect the equations 1, 2 and 3 and find that almost only the dynamic terms are affected by the void frac-

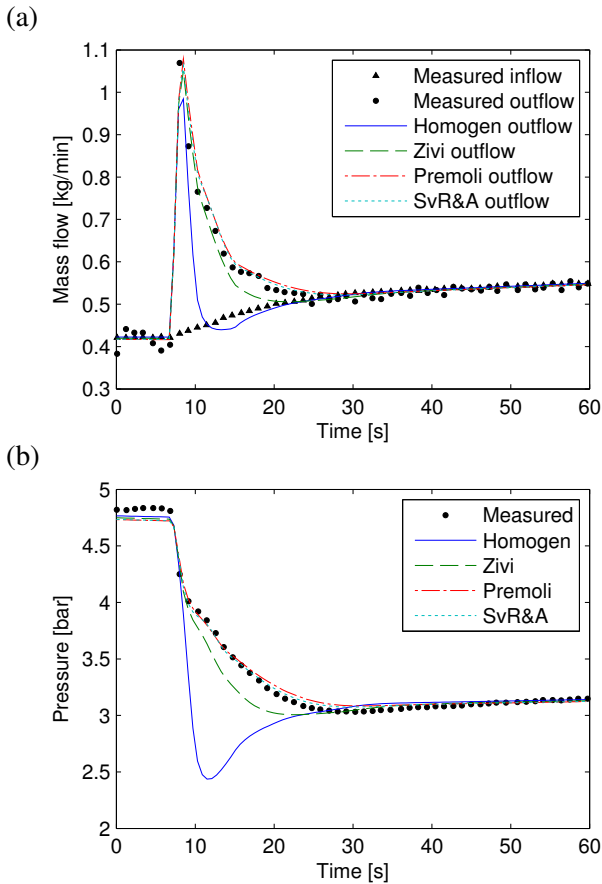


Figure 8: Mass outflow (a) and pressure outlet (b) response on increase in volume outflow

tion model. Other terms that are affected by the choice of void fraction model are the accelerational and gravitational pressure drop terms. For horizontal flow gravitational pressure drop vanish, however, usually both these terms are approximately an order of magnitude less than the frictional pressure drop [15, 28]. Furthermore, they do not influence the heat transfer and corresponding evaporation pressure.

The refrigerant charge in the evaporator (both two-phase and superheated area) are shown in table 2 at time equals 0 from figure 7 and 8, i.e. the two different steady states.

Table 2: Refrigerant charge in the evaporator at time=0 from figure 7 and 8

	Homogen [g]	Zivi [g]	Premoli [g]	SvR&A [g]
ss* (figure 7)	10.7	16.8	24.0	22.6
ss* (figure 8)	26.3	57.9	79.8	76.2

* ss = steady state (at time=0)

The two-phase area was approximately 35% of the

evaporator at time=0 from figure 7, however, on figure 8 at time=0 the two-phase area was approximately 85% in the evaporator. This leads to the differences in the refrigerant charge predictions from the steady states in figure 7 to 8 at time=0. When the volume flow goes down the pressure increases, both the overall UA-value and temperature difference between the refrigerant and the brine decreases, and it results in a smaller heat transfer and larger two-phase area. In other words, the refrigerant will be sucked out of the evaporator as the volume outflow increases at no control of the superheat.

Using the Premoli model and the Steiners version of the Rouhani and Axelsson model, as the most accurate void fraction models, shows that the homogeneous model underpredicts the refrigerant charge by approximately 2-3 times. The Zivi model lies in between.

6 Conclusion

It can be concluded that the homogeneous model is insufficient for modeling of the dynamic evaporator response of the current coaxial evaporator with high accuracy. If one wants to investigate the dynamic behavior due to refrigerant movement and amount of refrigerant in the evaporator, then a slip flow model is needed, since the homogeneous model gives inaccurate results. The choice of a given slip flow model must be considered for both numerical and accuracy reasons, which unfortunately are counteracting.

In this study the Premoli et al. [3] void fraction model and the Steiners version of the Rouhani and Axelsson [7] void fraction model gave the most accurate evaporator response. The simple void fraction model by Zivi [9] gave less accurate results, but quite much better than the homogeneous void fraction model.

These investigations considered a step in volume outflow, but similar conclusions with regard to void-fraction model validity are expected for a step in mass inflow to the evaporator, as pointed out in [10, 12]. Investigation of the control strategy of a capacity controlled compressor or an expansion valve using the measured superheat as feedback, are examples, where the use of a slip flow model is required.

References

- [1] M. A. Woldesemayat, A. J. Ghajar, Comparison of void fraction correlations for different flow patterns in horizontal and upward inclined pipes,

- International Journal of Multiphase Flow 33 (4) (2007) 347 – 370.
- [2] G. E. Dix, Vapor void fraction for forced convection with subcooled boiling at low flow rates, Tech. rep., General Electric Company Report NEDO-10491 (1971).
- [3] A. Premoli, D. D. Francesco, A. Prina, A dimensional correlation for evaluating two-phase mixture density, *La Termotecnica* 25 (1971) 17–26.
- [4] X. Maa, G. Dinga, P. Zhanga, W. Hana, S. Kasaharab, T. Yamaguchib, Experimental validation of void fraction models for r410a air conditioners, *International Journal of Refrigeration* 32 (2009) 780–790.
- [5] G. Mader, L. F. S. Larsen, G. P. F. Fösel, Low charge system behavior - interactions of heat exchanger volumes and charge, in: 2nd Workshop on Refrigerant Charge Reduction, IIR, KTH, Stockholm, Sweden, 2010.
- [6] L. Wojtan, T. Ursenbacher, J. R. Thome, Measurement of dynamic void fractions in stratified types of flow, *Experimental Thermal and Fluid Science* 29 (3) (2005) 383 – 392.
- [7] D. Steiner, Heat transfer to boiling saturated liquids, *VDI-Wärmeatlas (VDI Heat Atlas)*, Verein Deutscher Ingenieure (Ed.), VDI-Gesellschaft Verfahrenstechnik und Chemie-ingenieurwesen (GCV), Düsseldorf, 1993, (Translator: J.W. Fullarton).
- [8] L. Wojtan, T. Ursenbacher, J. R. Thome, Investigation of flow boiling in horizontal tubes: Part i - a new diabatic two-phase flow pattern map, *International Journal of Heat and Mass Transfer* 48 (2005) 2955–2969.
- [9] S. M. Zivi, Estimation of steady-state steam void-fraction by means of the principle of minimum entropy production, *J. Heat Transf.* 86 (1964) 247–252.
- [10] J. Antonius, Distribuerede fordampermodeller på flere detaljeringsniveauer, Master's thesis, Technical University of Denmark, Department of Energy Engineering (1998).
- [11] Cullimore & Ring Technologies Inc., Littleton, Colorado, USA, Sinda/Fluint user's manual, General Purpose Thermal/Fluid Network Analyzer, version 5.2 (2008).
- [12] A. Jakobsen, J. Antonius, H. J. Høgaard Knudsen, Experimental evaluation of the use of homogeneous and slip-flow two-phase dynamic models in evaporator modelling, in: 20th International Congress of Refrigeration, IIR/IIF, Sydney, 1999.
- [13] Dynasim AB, Research Park Ideon SE-223 70, Lund, Sweden, Dynamic Modeling Laboratory, Dymola User's Manual, version 7.4 (2010).
- [14] J. Eborn, H. Tummescheit, K. Prölb, Airconditioning - a modelica library for dynamic simulation of ac systems, in: 4th International Modelica Conference, Hamburg, Germany, 2005, pp. 185–192.
- [15] 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).
- [16] 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).
- [17] VDI Wärmeatlas, Berechnungsblätter für den Wärmeübergang, Springer-Verlag, Ch. Lab., 9th Edition, Dd 20, (2002).
- [18] S. M. Ghiaasiaan, Two-phase flow: Boiling and Condensation in Conventional and Miniature Systems, 1st Edition, Cambridge University Press, 2008.
- [19] S. V. Patankar, Numerical heat transfer and fluid flow, Taylor & Francis, 1980.
- [20] O. Bauer, Modelling of two-phase flows with modelica, Master's thesis, Lund University, Department of Automatic Control (1999).
- [21] A. F. Mills, Heat Transfer, 2nd Edition, Prentice Hall, 1999.
- [22] E. J. Dittus, L. M. K. Boelter, Publications on Engineering, Vol. 2, University of California, Berkeley, 1930.
- [23] V. Gnielinski, New equation for heat and mass transfer in turbulent pipe and channel flow, *International Chemical Engineering* 16 (1976) 359–368.

- [24] P. R. H. Blasius, VDI Wärmeatlas, 9th Edition, Springer-Verlag, Ch. Lab., 2002.
- [25] M. M. Shah, Chart correlation for saturated boiling heat transfer: Equations and further study, ASHRAE Transactions 88 (1982) 185–196.
- [26] H. Müller-Steinhagen, K. Heck, A simple friction pressure drop correlation for two-phase flow in pipes, Chemical engineering and processing 20 (1986) 297–308.
- [27] L. Wojtan, T. Ursenbacher, J. R. Thome, Investigation of flow boiling in horizontal tubes: Part ii - development of a new heat transfer model for stratified-wavy, dryout and mist flow regimes, International Journal of Heat and Mass Transfer 48 (2005) 2970–2985.
- [28] H. Jiang, Development of a simulation and optimization tool for heat-exchanger design, Ph.D. thesis, University of Maryland at College Park, Department of Mechanical Engineering (2003).

Strong Coupling of Modelica System-Level Models with Detailed CFD Models for Transient Simulation of Hydraulic Components in their Surrounding Environment

Antoine Viel
LMS Imagine
7 Place des Minimes, 42300 Roanne, France
antoine.viel@lmsintl.com

Abstract

Strong coupling with a CFD software is usually suited to the coupled transient simulation of an hydraulic component (like a valve, a pump, ...) with its surrounding environment. Due to the nature of the solvers used by CFD code, co-simulation is generally the best way to couple a Modelica system-level simulator and a CFD solver. This article describes a methodology and the associated technology for establishing a co-simulation between a Modelica model simulated with an ODE/DAE solver like the one encountered in LMS Imagine.Lab AMESim, and a 3D model of flow computed by a CFD software. The physical, numerical, and computer-related aspects of co-simulation handled by this methodology are exemplified on an application test case in fluid power.

Keywords: Tool Coupling; Co-simulation; Hydraulic Component Modeling; CFD

1 Introduction

The detailed design of an hydraulic component like a compound relief valve [1] is a complex task and requires a model including many details. The methodology to model this component implies starting with a simple model and after analysis and comparison with experimental results, the model is made more complex step by step. With this approach it is possible to understand the influence of different parts of the valve on the overall system transient behavior. However experimental results might not be available for all subsystems. In such cases involving complex 2D or 3D geometries with turbulent flows and distributed phenomena like cavitation, the use of Computational Fluid Dynamics simulation is of great help. CFD modeling is easy to setup when the fluid domain to be studied can

be decoupled from its surrounding environment. In such a case, the boundary conditions are well-known, and act as real sources with no reactions. If the flow is coupled to its environment, and this is frequently the case when performing transient simulation of a complete hydraulic circuit, the whole coupled system has to be simulated. This can be done by coupling the CFD model with the system-level model, the latter providing the boundary conditions of the meshed fluid domain. Coupling a system-level simulator with a CFD code can be performed through different approaches [7]:

- Weak coupling is well suited to the case of hydraulic models (or a part of it) which can be reduced to a static relationship between a small number of lumped variables. This relationship is usually characterized by performing batch runs in the CFD code, and gathering the results in lookup tables.
- Strong coupling with a CFD software is usually suited to the coupled transient simulation of an hydraulic component with its surrounding environment (remaining part of the hydraulic circuit, thermal exchanges, mechanical work done by the fluid, ...). Due to the nature of the solvers used by CFD code, co-simulation is generally the only way to couple a Modelica system-level simulator and a CFD solver [10].

The following sections describes a methodology for establishing a co-simulation between a Modelica simulation environment with an ODE/DAE solver and a 2D/3D model of flow computed by a CFD software. The physical, numerical, and computer-related aspects of co-simulation handled by this methodology are exemplified step-by-step on an application test case.

2 Application test case: modelling of a compound relief valve

2.1 Overall system description

The hydraulic system considered as test case is sketched out on Figure 1. This type of pilot-operated valve is usually used for limiting the pressure in an hydraulic circuit, by releasing a fraction of the flow from the inlet of the valve to its outlet. On the contrary of the direct-acting valves like the check-valves, operation of a compound relief valve is not affected by the flow going through it. A small pressure difference across the ball valve is enough to open the pilot valve independently of the main flow rate from inlet to outlet. The corresponding Modelica system-level model is shown on Figure 2.

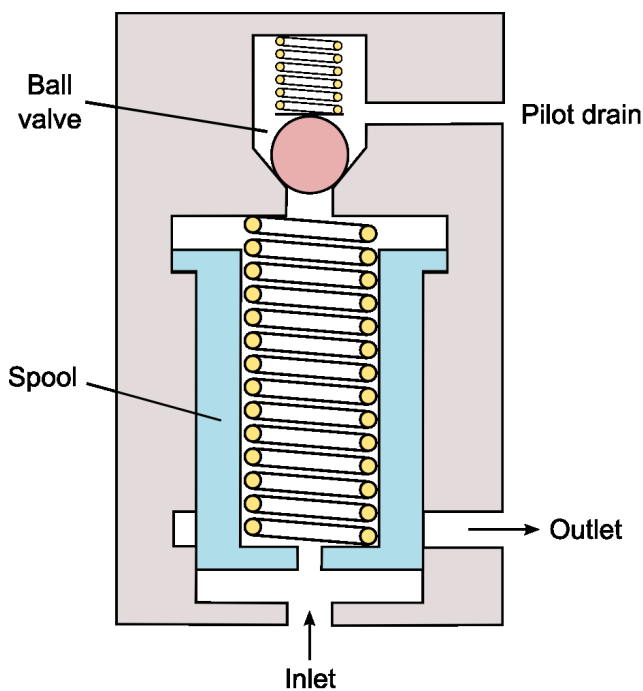


Figure 1: Sketch of the compound relief valve

The model tries to reproduce the transient behavior of the valve operating in a simplified hydraulic circuit made of a varying-flowrate pump and a load, on the left and right sides of the circuit sketch. At the center of the sketch, each component of the valve has a functional modeling counterpart on the Modelica model, using the hydraulic component design approach [2] [3]. The spool valve model is made of two parts:

1. The hydraulic part deals with the flow through the outer orifices of the spool, which areas depend on the spool position. It also describes the pressure dynamics in the top and bottom spool chambers

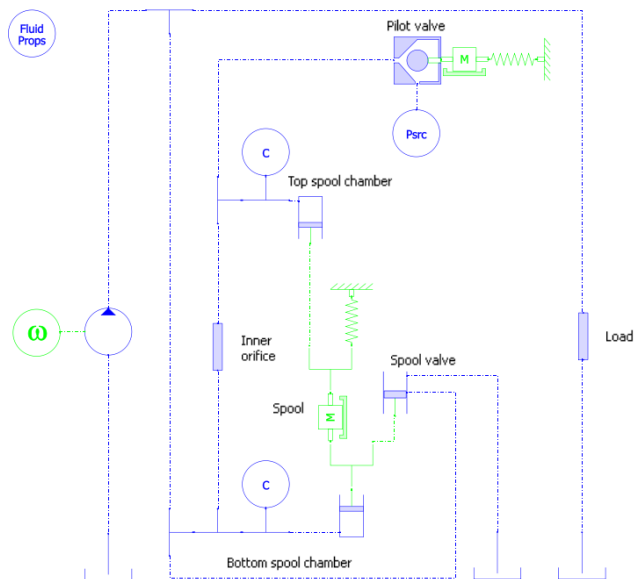


Figure 2: Sketch of the Modelica model of a compound relief valve.

using two hydraulic capacitive elements, which are explicitly materialized on the sketch. The corresponding pressures are acting on the spool body through the two piston-like elements.

2. The mechanical part directly represents the rigid body dynamics of the spool submitted to the pressure, spring and viscous damping forces.

At the top of the model sketch lies the pilot valve, which is a ball poppet valve. The hydraulic part of this model is aimed at being replaced by the detailed CFD model shown on Figure 3. A simplified Modelica model of the flow in the valve seat is described in the next section. The remaining mechanical part is the rigid body dynamics of the ball submitted to the static and dynamic fluid forces and to the spring force. In the model, the motion of the ball is limited in its travel by an ideal endstop which corresponds to the pilot valve being closed.

2.2 System-level model of the pilot valve

To study the stability of the coupled system resulting from the co-simulation of the detailed CFD model of the ball valve with the system-level of the hydraulic circuit and mechanical part of the valve, a simplified equivalent system-level model of the ball valve is needed.

The model tries to reproduce the transient behavior of the ball poppet submitted to the pressure and viscosity forces arising from the turbulent flow in the valve

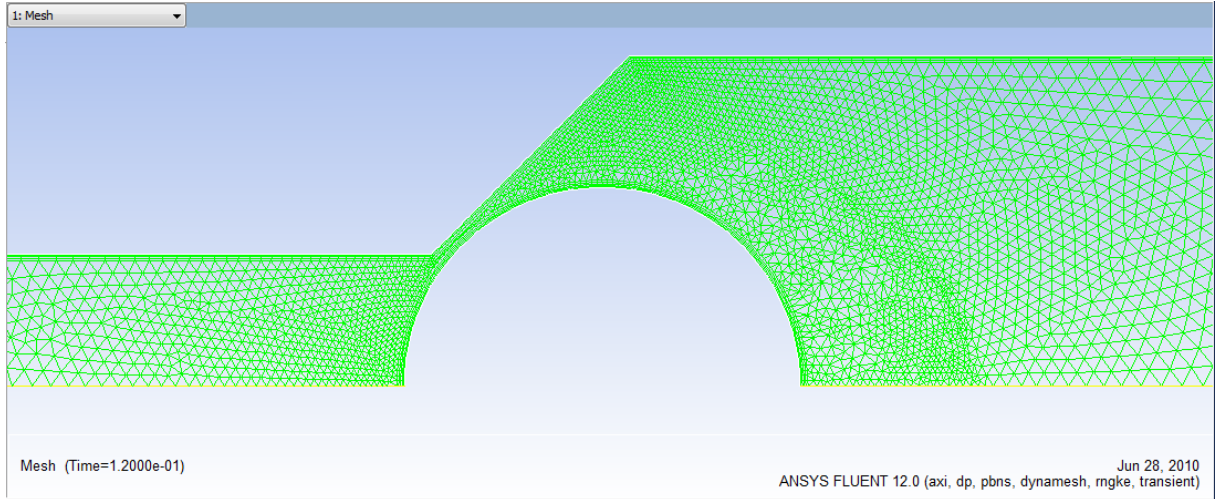


Figure 3: 2D axisymmetric fluid domain considered for detailed flow modeling.

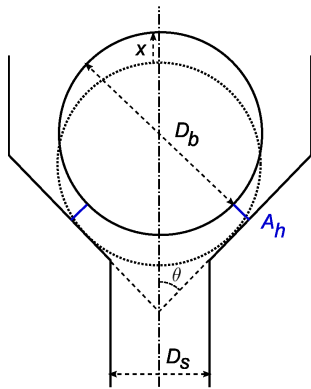


Figure 4: Geometry of the ball poppet valve.

seat. The modeling assumptions and analysis follow roughly [4]. The flow rate through the valve is given by the elementary orifice law:

$$q = c_d A_h(x) \sqrt{\frac{2(p_{up} - p_{dn})}{\rho(\bar{p})}} \quad (1)$$

where p_{up} , p_{dn} , and $\bar{p} = \frac{1}{2}(p_{up} + p_{dn})$ are the upstream, downstream and mean pressures, ρ the fluid density depending mainly on pressure, A_h the throat area, and c_d the discharge coefficient. The throat area, shown on Figure 4 depends geometrically on the ball lift x :

$$A_h(x) = \frac{\pi}{2} x \sin \theta (x \sin \theta + D_b) \quad (2)$$

with D_b the ball diameter and θ the half chamfer angle. The discharge coefficient is usually modeled as a smooth function of the flow number:

$$c_d = c_{d,max} \tanh\left(\frac{2\lambda}{\lambda_{crit}}\right)$$

the flow number λ being given by:

$$\lambda = \frac{D_h}{\mu(\bar{p})} \sqrt{2\rho(\bar{p})(p_{up} - p_{dn})}$$

λ_{crit} is the critical value of the flow number, corresponding to the laminar-turbulent transition. The hydraulic diameter D_h is directly linked to the ball lift by a geometrical relation:

$$D_h = 2x \sin \theta \quad (3)$$

When the ball is rising, the actual flow rate is bounded by the inlet area rather than the throat area. This is taken into account in the Modelica model by computing a maximum lift x_{max} such that in equation (2) the area becomes equal to the inlet area:

$$A_h(x_{max}) = \frac{\pi}{4} D_s^2$$

where D_s is the seat diameter. The effective lift value, bounded by x_{max} is then used to compute the throat area (2) and the hydraulic diameter (3).

The hydromechanical part of the model deals with the fluid forces acting on the ball. Static pressure forces are usually distinguished from dynamic forces stemming from the acceleration of fluid through the orifice. Under varying flow conditions resulting from varying lift, the pressure distribution along the ball can have large variations. The dependency of the pressure distribution, and hence of the active area used in the computation of the static pressures forces, could be studied using a preliminary *uncoupled* CFD computation [6]. The pressure field is computed at steady-state for different value of the ball lift. Some results are shown on Figure 5.

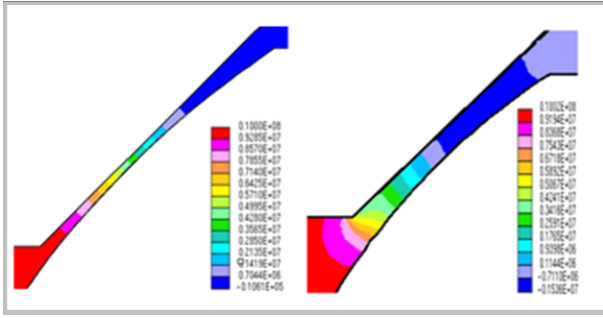


Figure 5: Pressure distribution in the orifice for two different fixed ball lifts.

For laminar flows, the pressure drop occurs at the minimum geometrical cross section

$$A_l = \frac{\pi}{4} D_b^2 \cos^2 \theta$$

As it is shown on Figure 5 when transiting from small ball lift to high lift, the flow becomes turbulent and separates from the ball, and the corresponding pressure distribution moves towards the inlet [5]. A model, proposed by [4], takes this into account by considering the laminar-turbulent transition. The active area A_p is thus a fraction of the geometrical pressurized area:

$$\frac{A_p}{A_l} = \begin{cases} 1 + \sqrt{\frac{\lambda}{\lambda_{crit}}} (f - 1) & \text{if } \lambda \leq \lambda_{crit} \\ f & \text{if } \lambda > \lambda_{crit} \end{cases} \quad (4)$$

where $0 < f < 1$ is a turbulent active area factor, fitted on the steady-state CFD computations. Finally, the fluid forces acting on the ball are given by the sum of the static pressure forces and the dynamic or jet forces:

$$F = A_p (p_{up} - p_{dn}) + F_{jet}$$

with

$$\begin{aligned} F_{jet} &= \rho(\bar{p}) q (v_{up} - v_{dn} \cos \theta) \\ &= 2 c_d A_h(x) (p_{up} - p_{dn}) \left(c_d \frac{A_h(x)}{A_h(x_{max})} - \cos \theta \right) \end{aligned}$$

3 Coupling principles and methodology

When trying to couple simulators, many different issues arise (types of physical coupling, numerical methods, software and hardware implementations) which seem intricate at a first glance. A general methodology [7] is required to prioritize these issues, thus avoiding suboptimal choices based only on computer-related contingencies. The decision flow

chart shown on Figure 6 synthesizes this methodology used to couple a system-level simulator with third-party simulation software. The choices made for coupling the 1D system-level model of an hydromechanic system with a CFD model are emphasized on the flow chart. The next sections explain these choices.

3.1 Type of physical coupling

The first choice to be made is between weak coupling or strong coupling. Weak coupling means that the model could be reduced to a dynamic part, cascaded with a non-linear static part, obtained by statically characterizing the system on some operating points, typically using some batch run functionality of the CFD software. Weak coupling has many advantages from the numerical and the software point of view: the external model is generally reduced to data tables that are evaluated directly in the system-level simulator, using a table lookup library [7]. However, as we are interested in the study of the coupled system in general transient conditions, the weak coupling seems inappropriate. Static characterization - if it makes sense - would require a lot of batch run computations with varying boundary conditions. As the model reduction assumptions cannot be always made about a CFD model, strong or full coupling between the two models is considered here.

3.2 Solver interaction

The two models being strongly coupled, the next choice to be done deals with the numerical solvers used to perform the transient simulation of the coupled system. The best way is to use only one numerical integration solver, simulating a unique mathematical model, obtained by importing one of the two involved model into the other one. This way, the numerical issues are only related to the capabilities of the numerical integration scheme to handle the physically-originating stiffnesses of the coupled system, which could typically be handled by variable-timestep variable-order linear multistep methods. Importing the external model into the system-level simulator implies obviously that this model could be exported from its simulation environment, either the mathematical structure of the equations, or the value of the parameters to be fed to an equivalent model. However, with most of the CFD software [9], the mathematical structure of the fluid model (including conservation laws like the Navier-Stokes equations and constitutive relations like fluid properties or rheological

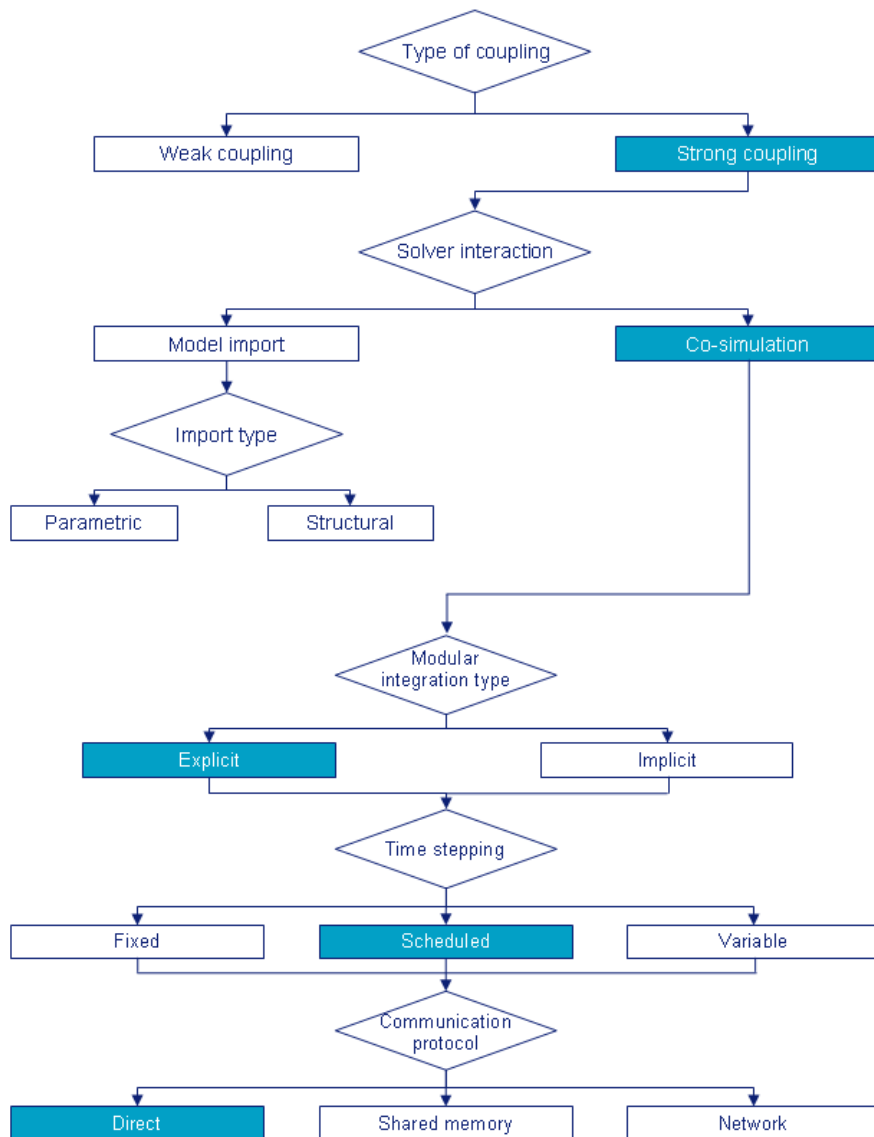


Figure 6: Decision flow chart for a general simulators coupling methodology.

models) is tightly linked with the numerical methods used to discretize it. Exporting the whole fluid model from a CFD software seems very difficult to realize, and the solver capabilities that would be required to solve such a model are very specific, and are not available in general system-level simulators. That is why the two solvers must be retained, and the transient simulation is performed through co-simulation.

3.3 System partitioning for co-simulation

In co-simulation, the two involved solvers exchange only a predefined set of variables at some communication time point. Thus, the whole coupled system must be partitioned in two subsystems, and the exchanged variables on the boundary have to be precisely defined.

The Modelica model described on Figure 2 is modified in order to delegate the detailed modelling of the flow in the seat to the CFD code. The modified part of the AMESim-Modelica model is shown on Figure 7. A non-standard construction called *external connector* is introduced to enforce the causality at the boundary of the Modelica model. A Modelica model with such connectors is processed by the AMESim Modelica compiler in the usual a-causal way, except for the variables declared as external inputs which have their causality imposed by the outside world. The part of the model corresponding to the valve seat is removed from model and three causal/a-causal gateway models are introduced at the boundary of the model:

```

external connector ConnectForceInPosOut
output SIunits.Velocity v;
output SIunits.Position x;
    
```

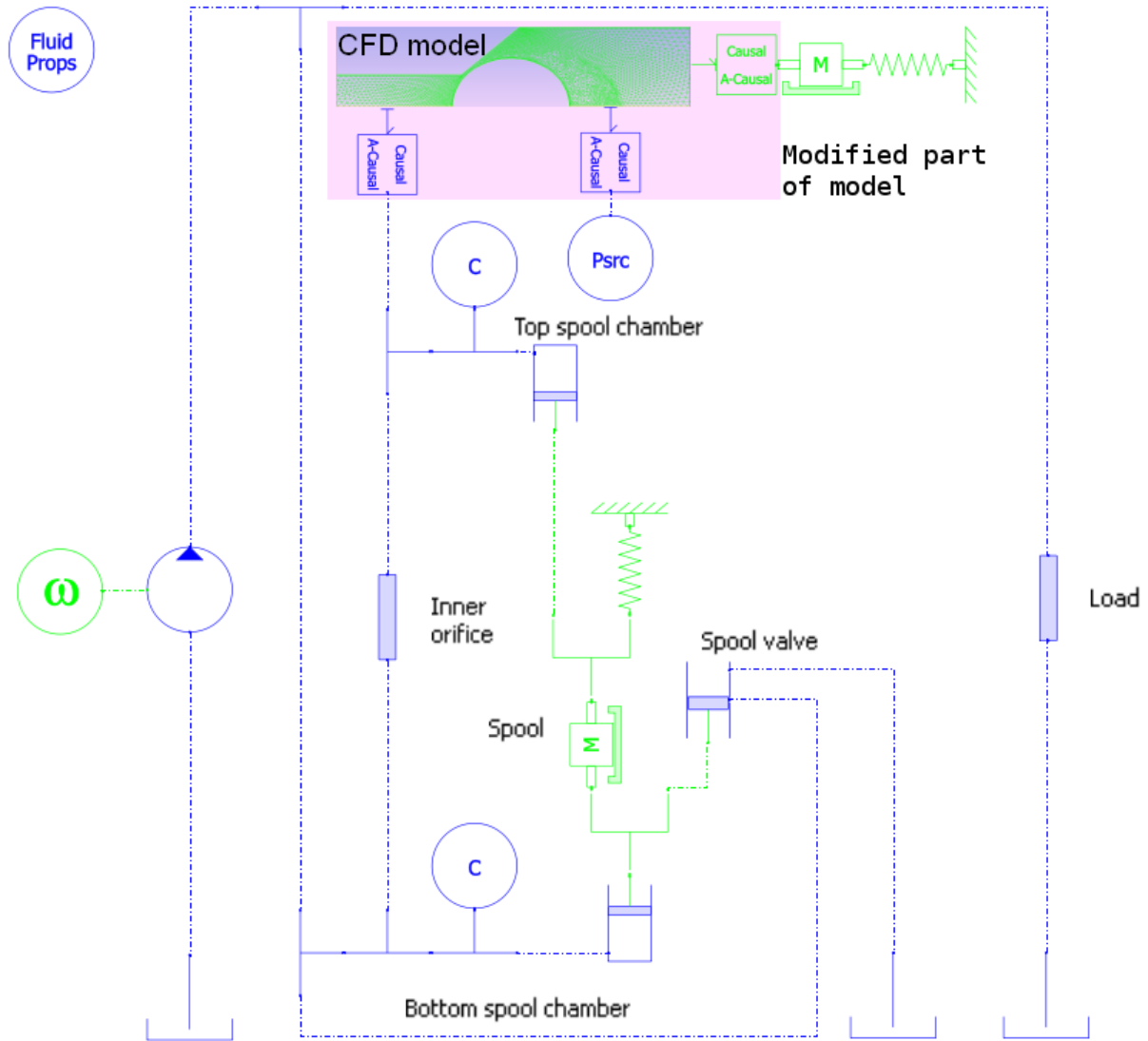


Figure 7: Modified Modelica model with external connectors.

```

input SIunits.Force f;
end ConnectForceInPosOut;

model ForceInputPosOutput
  ConnectForceInPosOut c "Causal connector";
  Flange a "A-causal connector";
equation
  a.f + c.f = 0;
  a.x = c.x;
  a.v = c.v;
end ForceInputPosOutput;

external connector ConnectFlowInPressureOut
  output SIunits.Pressure_bar p;
  input SIunits.VolumeflowRate q;
end ConnectFlowInPressureOut;

model FlowrateInputPressureOutput
  ConnectFlowInPressureOut c "Causal connector";
  FluidPort fp "A-causal connector";
equation
  fp.q + c.q = 0;
  fp.p = SIunits.from_bar(c.p);
end FlowrateInputPressureOutput;

```

These models act as sources or sensors from the system-level point of view and they carry the exchanged variables shown on Table 1.

In the system-level to CFD direction, the variable is directly read from the Modelica model, and applied as a space-constant boundary condition (pressure, wall position and velocity) to the CFD model. In the other direction, the variable is computed by integrating the related quantity (pressure giving force, fluid velocity giving flow rate) on the boundaries. Some additional unit conversions are performed in the external causal/a-causal connectors, since the Modelica library upon which the model is built works with SI units, whereas the CFD software user functions used for specifying the boundary conditions could be written in trade units.

Modelica Model	Variable	Causality
ForceInputPosOutput	x (ball position)	to CFD
ForceInputPosOutput	v (ball velocity)	to CFD
ForceInputPosOutput	f (fluid forces acting on ball)	from CFD
FlowrateInput PressureOutput (instance 1)	p (pressure at inlet)	to CFD
FlowrateInput PressureOutput (instance 1)	q (flow rate at inlet)	from CFD
FlowrateInput PressureOutput (instance 2)	p (pressure at outlet)	to CFD
FlowrateInput PressureOutput (instance 2)	q (flow rate at outlet)	from CFD

Table 1: Variables exchanged during co-simulation.

3.4 Modular integration type

The two subsystems being coupled through the state variables listed on Table 1, there is no need to set up a full algebraic coupling method, and hence only the explicit modular integration method is implemented on each simulator, as depicted on Figure 8. The simulation time is partitioned in macro time step, in which the integration process is strictly cascaded from a simulator to another, whatever the actual order. Inside a macro time step, the exchanged variables are held constant.

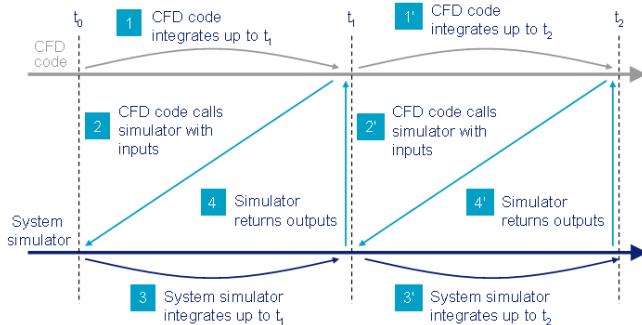


Figure 8: Explicit modular integration scheme.

3.5 Time stepping

The choice of a co-simulation time step T (or macro time step size) is determined by numerical stability of the co-simulated system with respect to the continuous case. Co-simulating two systems with explicit modular integration method introduces a sample with zero-order hold on the exchanged variables. The whole system obtained is a loop sampled system, which numerical stability may differ from the intrinsic physical stability of the fully coupled continuous system. Stability study of such loop sampled system is carried using the usual stability criteria from linear control system theory [8]. The coupled system comprising the seat and the mechanical part of the valve has many non-linearities, arising from geometry or pressure-dependent fluid properties. To study the stability of the loop sampled system, the system has to be linearized around some operating point. This can be performed using the Linear Analysis Tools from LMS Imagine.Lab, once the most dimensioning operating point has been recognised. At high ball lift, the flow in the seat becomes turbulent, and flow separation occurs. This means that the pressure drop moves towards the inlet, and the active area of the ball is decreased, while the influence of the momentum forces is increased. Stability study of the coupled system is thus carried by considering a linearized model of the mechanical part of the valve subjected to the sole jet or momentum forces of the fluid. In this simplified linearized system, co-simulation is taken into account by considering a sample and zero-order hold at rate T of the ball position. The hydraulic stiffness k_h represents the sensitivity of the jet forces with respect to the valve lift, at the desired operating point (typically high lift). The bloc-diagram of the resulting loop sampled system is depicted on Figure 9.

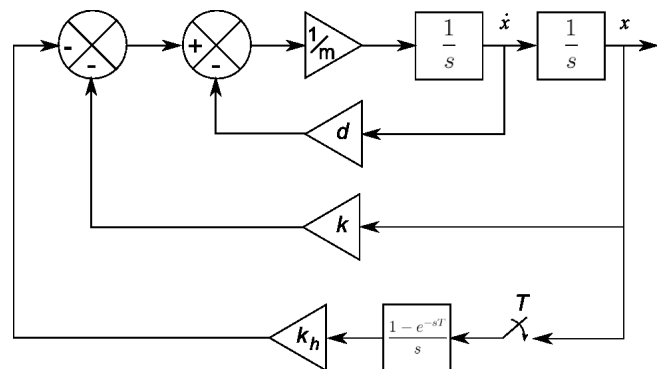


Figure 9: Bloc diagram of the loop sampled system considered for studying co-simulation stability.

The discrete transfer function of this loop sampled system is then given by:

$$\begin{aligned} H_{loop}(z) &= \mathbf{Z} \left[\frac{1}{ms^2 + ds + k} \frac{1 - e^{-Ts}}{s} k_h \right] \\ &= (1 - z^{-1}) K_{loop} \mathbf{Z} \left[\frac{\omega_0^2}{s(s^2 + 2\zeta\omega_0s + \omega_0^2)} \right] \end{aligned}$$

with $K_{loop} = \frac{k_h}{m\omega_0^2}$ the loop gain, $\omega_0^2 = \frac{k}{m}$ the undamped natural frequency, and $\zeta = \frac{d}{2\sqrt{km}}$ the mechanical damping ratio. Finally the loop transfer function is given by:

$$H_{loop}(z) = \frac{K_{loop}(az + b)}{z^2 - 2ze^{-\zeta\omega_0T} \cos \omega_d T + e^{-2\zeta\omega_0T}} \quad (5)$$

with $\omega_d = \sqrt{1 - \zeta^2} \omega_0$,

$$a = 1 - e^{-\zeta\omega_0T} \cos \omega_d T - \frac{\zeta e^{-\zeta\omega_0T}}{\sqrt{1 - \zeta^2}} \sin \omega_d T$$

and

$$b = e^{-2\zeta\omega_0T} - e^{-\zeta\omega_0T} \cos \omega_d T + \frac{\zeta e^{-\zeta\omega_0T}}{\sqrt{1 - \zeta^2}} \sin \omega_d T$$

At the operating point corresponding to high ball lift, the Linear Analysis Tool of LMS Imagine.Lab gives the following values for the loop transfer function coefficients: $K_{loop} = 3.85$, $\frac{\omega_0}{2\pi} = 50$ Hz and $\zeta = 0.47$. For a co-simulation macro-time step $T = 0.1\mu\text{s}$, the Bode diagram of H_{loop} is shown on Figure 10. Applying the Nyquist stability criterion to (5) with these values, the crossover frequency is $f_{cross} = 390$ Hz, giving a gain margin value $GM = 10 \log_{10}(|H_{loop}(f_{cross})|) = 12$ dB, and a phase margin $PM = 28^\circ$. These values are enough for ensuring the numerical stability of co-simulation, and hence the time step size may be scheduled up to $0.1\mu\text{s}$ at each macro step.

4 Communication protocol and computer implementation

The last choice to be done is related to the communication protocol implemented between the two simulators. Computational performance is usually considered for choosing the communication channel. Distributed co-simulation (two software running on two different processors or two different computers) leading to parallel processing speedup is interesting only if computational load is well balanced between the two software. This is clearly not the case when co-simulating a system-level model with a CFD code.

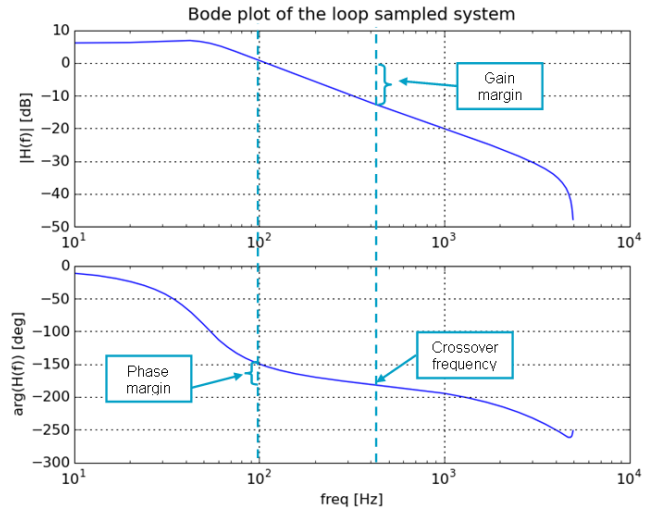


Figure 10: Bode diagram of the loop transfer function for a co-simulation time step $T = 0.1\mu\text{s}$.

The load is strongly unbalanced: typically, the complete transient simulation of the step response with the system-level model takes less than 1 minute, whereas the valve simulated with the detailed CFD model of the flow in the seat takes about 1 day on a quad core CPU computer. Distributed co-simulation being useless, there is no need for multiprocessor or multicommputer communication protocol. A direct local communication link is therefore used on the same processor. The Modelica model with embedded LMS Imagine.Lab AMESim solver and Modelica libraries is compiled as a shared library using the Generic Co-simulation Interface of LMS Imagine.Lab, and then is linked locally with the CFD software, namely ANSYS Fluent. At each macro time step, ANSYS Fluent is acting as the master simulator: it schedules the next macro step size, and calls the AMESim solver with the input variables, according to the modular integration scheme depicted on Figure 8. Output variables are exchanged at the end of the current macro time step, and a new macro step can take place.

5 Some co-simulation results

As an example, we consider computing the step response of the check valve, by increasing the pressure at inlet from 0 to 10 bar in 1 ms. Some of the exchanged variables (ball lift, hydraulic forces, upstream pressure, flow rate) are monitored in LMS Imagine.Lab during co-simulation (Figure 11), while the other flow quantities - static pressure, velocity, Reynolds number, etc - are displayed directly in the CFD software (Figure 12).

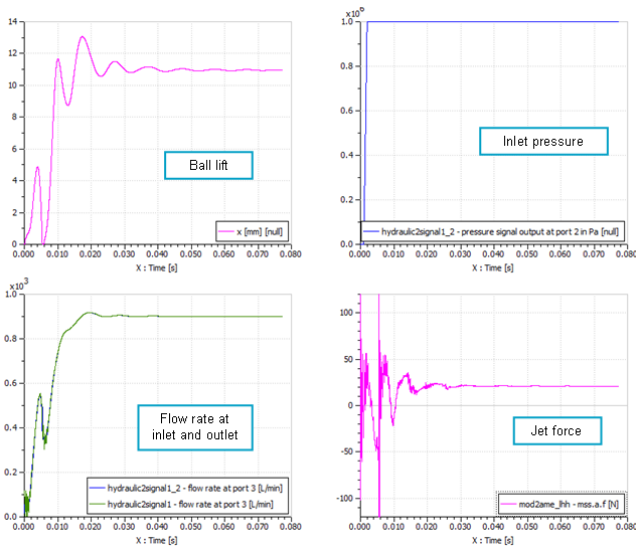


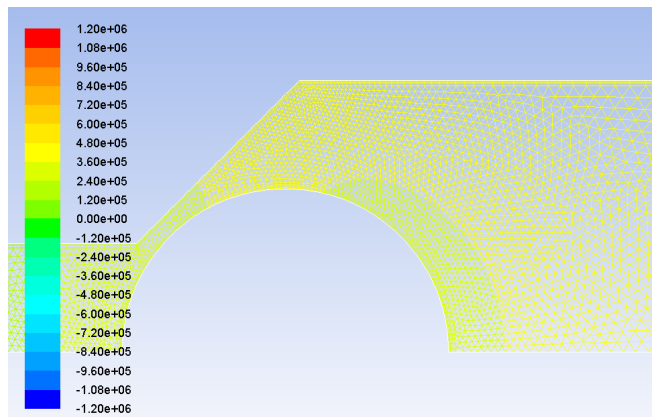
Figure 11: Exchanged variables monitored in LMS Imagine.Lab

The variables monitored in LMS Imagine.Lab AMESim are the typical responses to such an input of pressure. It is interesting to note that the jet force values (i.e. hydrodynamic forces) can be reached, since it is a key factor in the design of such and hydro-mechanical systems.

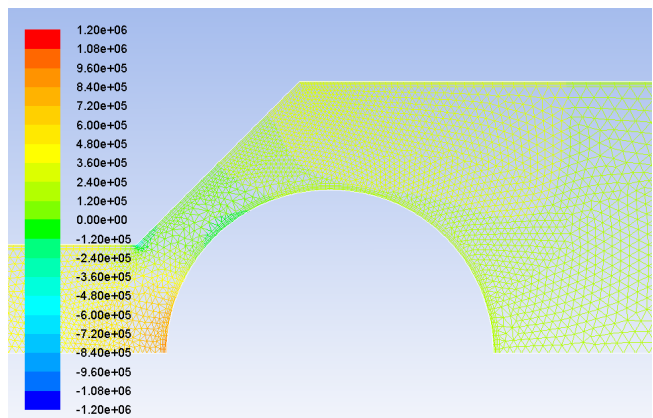
The map of the static pressure can be represented for the whole geometry in the CFD software. It allows accessing detailed results that can not be reached in usual lumped-parameter models. The map of static pressure obtained here at different simulation times corresponds to the expected results for such a use-case of poppet with ball on conical seat [4].

6 Conclusions

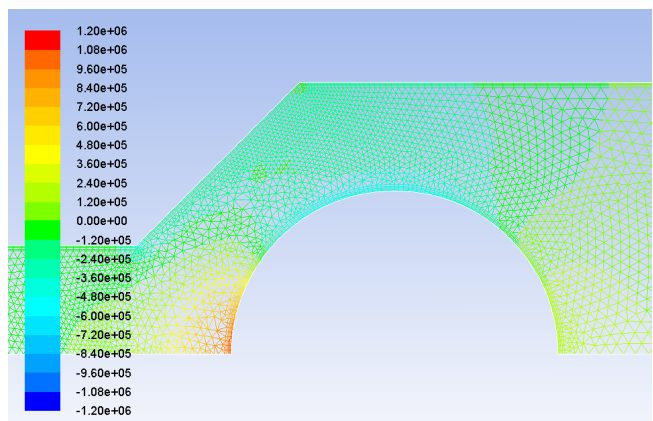
We can finally conclude that the strong coupling of some Modelica components with a CFD model combining two different software such as LMS Imagine.Lab AMESim for the system-level tool and ANSYS Fluent for the CFD software was successful. Beside the fluid power test case exemplified in this paper, the methodology was also applied to the modeling of a full direct diesel injection system [10] using the 3D CFD code Principia Eole to accurately predict the cavitation transients in a nozzle under multiple injection conditions. It would require some additional works to apply this methodology outside fluid power to confirm that it brings a real added-value for other types of industrial applications. Another promising way of research lies in the Functional Mock-up Interface (FMI, [11]), which standardises the coupling be-



Contours of Static Pressure (pascal) (Time=1.0100e-03) May 06, 2010
ANSYS FLUENT 12.0 (axi, dp, pbns, dynamesh, rngke, transient)



Contours of Static Pressure (pascal) (Time=4.2200e-03) May 06, 2010
ANSYS FLUENT 12.0 (axi, dp, pbns, dynamesh, rngke, transient)



Contours of Static Pressure (pascal) (Time=1.1010e-02) May 06, 2010
ANSYS FLUENT 12.0 (axi, dp, pbns, dynamesh, rngke, transient)

Figure 12: Pressure distributions at t = 1 ms, 4.2 ms and 11 ms.

tween two solvers through co-simulation. The general simulator coupling methodology described here could benefit from this FMI specifications, especially regarding the implementation of implicit modular integration and time stepping techniques by CFD software editors.

Acknowledgments

This work was in part supported by DGE in the ITEA2 Eurosyslib (06020) project under contract number 07.2.93.0146.

References

- [1] Akers A., Gassman M., Smith R., Hydraulic Power System Analysis, CRC Press, 2006.
- [2] Mc Cloy D., Martin H.R., Control of Fluid Power: Analysis and Design, Wiley, 1980.
- [3] Lebrun M., Richards C.W., How to create good models without writing a single line of code, 4th Scandinavian Int. Conference on Fluid Power, Linköping, Sweden, June 1997, vol. 1.
- [4] Mittwollen N., Michl T., Breit R., Parametric hydraulic valve model including transitional flow effects. In: Proceedings of the 2nd MATHMOD Vienna, IMACS Symposium on Mathematical Modelling, Febr. 5-7, 1997, TU Vienna, Austria, Editors: I. Troch, F. Breitenecker, ARGESIM Report No.11.
- [5] Clavier A., Alirand M., Vernat F., Sagot B., Local Approach to Improve the Global Approach of Hydraulic Forces in Ball Poppet Valves, 4th In. Symposium on Fluid Power, Wuhan, China, April 2003.
- [6] Baudry X., Mare J.C., Linking CFD and lumped parameters analysis for the design of flow compensated spool valve, 1st Fluid Power Net Int. PhD Symposium, Hamburg, Germany, June 2000.
- [7] Neyrat S., Viel A., Strong Coupling LMS Imagine.Lab Modelica with CFD Software, Report of the sub work package 2.4 of the ITEA2 Eurosyslib Project, LMS Imagine, June 2010.
- [8] Franklin G.F., Powell J.D., Workman M.L., Digital Control of Dynamic Systems, Addison Wesley, 1997.
- [9] Chung T.J., Computational Fluid Dynamics, Cambridge University Press, 2002.
- [10] Marcer, R., Audiffren C., Viel A., Bouvier B., Walbott A., Argueyrolles B., Coupling 1D System AMESim and 3D CFD EOLE models for Diesel Injection Simulation. In: Proceedings of the ILASS Europe 23rd Annual Conference on Liquid Atomization and Spray Systems, Brno, Sept. 2010.
- [11] Functional Mock-Up Interface for Co-Simulation 1.0 Specification, MODELISAR project (ITEA2 - 07006) and consortium, October 12th, 2010.

Recent Developments of the Modelica “Buildings” Library for Building Energy and Control Systems

Michael Wetter, Wangda Zuo, Thierry Stephane Noudui
 Simulation Research Group, Building Technologies Department,
 Environmental Energy Technologies Division, Lawrence Berkeley National Laboratory,
 Berkeley, CA 94720, USA

Abstract

At the Modelica 2009 conference, we introduced the **Buildings** library, a freely available Modelica library for building energy and control systems [16].

This paper reports the updates of the library and presents example applications for a range of heating, ventilation and air conditioning (HVAC) systems. Over the past two years, the library has been further developed. The number of HVAC components models has been doubled and various components have been revised to increase numerical robustness.

The paper starts with an overview of the library architecture and a description of the main packages. To demonstrate the features of the **Buildings** library, applications that include multizone airflow simulation as well as supervisory and local loop control of a variable air volume (VAV) system are briefly described. The paper closes with a discussion of the current development.

Keywords: *building energy systems, heating, ventilation, air-conditioning, controls*

1 Introduction

Buildings account for a large portion of energy consumption and related green house gas emissions. For example, in the United States, buildings consume 2/3 of electricity and 40% of total energy [4]. In order to reduce global green house gas emissions, it is critical to reduce building energy consumption by increasing energy-efficiency and by using more renewable energy. To support the design and operation of low energy buildings, a simulation program should support:

1. rapid prototyping of new building systems,
2. comparison of the performance of different designs of the building, its energy system and its

- control algorithms,
3. analysis of the operation of existing building systems,
4. development and specification of building control sequences, and
5. reuse of models during operation for energy-minimizing controls, fault detection and diagnostics.

To support these use cases, we develop an open-source Modelica library for building energy and control systems. The library is freely available from <http://www.modelica.org/libraries/Buildings>.

At the 7th Modelica conference in 2009, we introduced the **Buildings** library, version 0.6.0, which had 73 non-partial models and blocks, as well as 26 functions. The latest version, 0.10.0, has 129 non-partial models and blocks and 39 functions. This paper highlights some of these updates to inform users about the new capabilities.

The paper is structured as follows: Section 2 gives an overview of the **Buildings** library. Section 3 describes the updates in detail. Section 4 presents applications with models for multizone airflow simulation and for co-simulation. Section 5 describes classes which are currently under development and will be available in future releases.

2 Summary of the Buildings Library

The **Buildings** library is based on the **Modelica.Fluid** library [8]. The **Buildings** library is organized into the packages shown in Fig. 1. Components in these packages augment models from the Modelica Standard Library and from the **Modelica.Fluid** library. Base classes, which are typically not of interest to the end-user, but are used to construct other classes, are stored in packages called **BaseClasses**. Most packages

contain a package called **Examples**, which contains example applications. The examples illustrate typical use of components in the parent directories. They are also used to conduct unit tests.

The current **Buildings** library contains six major packages, including a new package **Airflow**. The package **Airflow** provides models to compute the airflow inside the buildings and between the building and the ambient environment. It currently has a package **Multizone** for multizone airflow models, which is discussed in Section 3.1. The **Controls** package contains components for continuous time control, discrete time control and scheduling of set points. The package **Fluid** is the largest package of the **Buildings** library. It contains components for fluid flow systems, such as pumps, boilers, chillers, valves and sensors, which are described in Section 3.3. The package **HeatTransfer** contains models for heat transfer in buildings (Section 3.4). It provides models and functions for heat transfer due to convection, conduction and radiation. It also has thermal property data for different solid materials. The **Media** package contains media models that are simpler and generally computationally more efficient than the ones in the Modelica Standard Library. The simplifications have been done by taking into account that HVAC systems in buildings typically have a smaller range of operating conditions compared to other thermodynamic applications. The **Utilities** package provides utilities such as for the calculation of thermal comfort and psychrometric properties. It also has an interface to the Building Control Virtual Test Bed (BCVTB) [17], which can connect Modelica to other simulation programs, such as EnergyPlus, MATLAB/Simulink and Radiance, for co-simulation. The BCVTB can also connect Modelica models to building automation systems for model-based operation.

3 Updates of the Buildings Library

This section compares the newest **Buildings** library, version 0.10.0, with version 0.6.0 which was reported in the last Modelica conference [16]. The purpose is to explain to users the new packages, models and blocks, as well as their new features.

3.1 Package Airflow

The airflow package provides models for computing air flow inside a building and between a building and its exterior environment. For build-

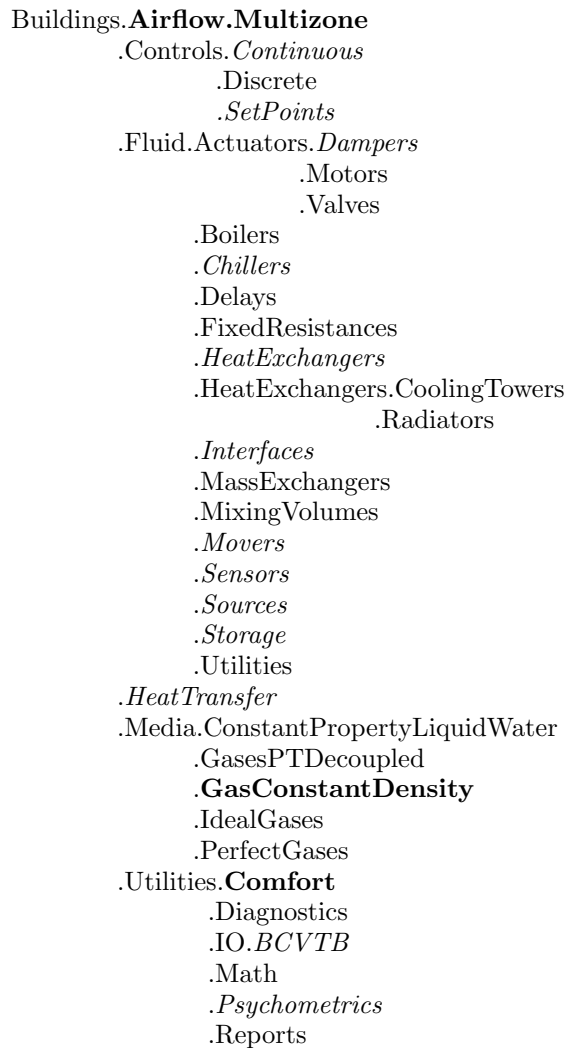


Figure 1: Package structure of the **Buildings** library. Only the major packages are shown. **Bold** indicates a new package and *italic* indicates existing packages with new models added.

ing simulation, many different indoor airflow models are generally used, such as multizone network models [3], zonal models [12], computational fluid dynamics [13], and fast fluid dynamics [18]. For indoor airflow simulation, multizone network models with the well-mixed air assumption are fast but not appropriate if the air is stratified. By solving the Navier-Stokes equations and equations for conservation of mass and energy, computational fluid dynamics (CFD) is the most detailed and accurate modeling method. However, computing time for CFD is large for flow simulation in a large building or over a long time horizon. To fill the gap between multizone and CFD, fast fluid dynamics models solve the Navier-Stokes equations with simplified schemes that are much faster but less accurate than CFD.

Currently, the `Airflow` package contains multizone airflow models in the package `Multizone`. These models compute the airflow and contaminant transport between different rooms, as well as between a room and the exterior. Multizone airflow models assume that air and contaminants in each room volume are completely mixed. The driving force for the air flow is pressure difference induced by flow imbalance of the HVAC system, density difference across large openings (such as open doors or windows), stack effects in high rise buildings, and wind pressure on the building facade.

The air volume in each room is modeled by an instantaneously mixed volume that provides differential equations for conservation of mass, species concentration, trace substances and internal energy. As in `Modelica.Fluid`, a parameter can be used to switch between steady-state and transient simulation, and to switch the initialization equations between steady-state initialization and prescribed state variables.

The flow resistance between these volumes is computed using the orifice equation

$$\dot{V} = C_d A \sqrt{2/\rho} \Delta P^m, \quad (1)$$

where \dot{V} is the volume flow rate, C_d is the dimensionless discharge coefficient, A is the cross section area of the opening, ρ is the density of the fluid, ΔP is the static pressure difference and m is the flow exponent. Large openings are characterized by m very close to 0.5, while values near 0.65 have been found for small crack-like openings. Typical values for C_d and m can be found in [15] and in the citations therein. For pressure differences that are smaller in magnitude than a user-specified parameter, equation (1) is regularized to ensure that it is differentiable with a continuous derivative.

The model `EffectiveAirLeakageArea` computes air leakage. It describes a one-directional pressure driven air flow through a crack-like opening. The opening is modeled as an orifice. The orifice area is parameterized by processing the effective air leakage area, the discharge coefficient and pressure drop at a reference condition. The effective air leakage area can be obtained, for example, from the ASHRAE fundamentals [1]. A similar model is also used in the multizone airflow modeling software CONTAM [5].

To compute the bi-directional flow across large openings, such as doors, the opening is discretized along its height into compartments. Then, the orifice equation (1) is used to compute the flow for each compartment as explained

in [15]. The model `DoorDiscretizedOpen` describes a door that is always open, and the model `DoorDiscretizedOperable` describes a door whose opening area can be changed using a control signal.

To model the pressure difference caused by stack effect, one can use the model `MediumColumn` for a steady-state and `MediumColumnDynamic` for a transient model. The model `MediumColumn` computes the pressure difference at its ports using

$$\Delta p = h \rho g, \quad (2)$$

where h is the height of the medium column, ρ is the density and g is the earth acceleration. The model `MediumColumn` can be parameterized to use for ρ the density of either port, or the density of the inflowing medium. The latter situation allows, for example, modeling of a vertical shaft, such as a chimney, whose density may be equal to the one of the inflowing medium. The model `MediumColumnDynamic` contains, in addition to (2), also a mixing volume that may be used to approximate the transient response of the medium column, or to inject heat into the air stream as may happen in a solar chimney in which walls absorb solar radiation and heat the fluid inside the chimney to increase the buoyancy force.

The models `ZonalFlow_ACS` and `ZonalFlow_m_flow` can be used to exchange a fixed flow rate between two volumes. As an input, they use the air exchange rate per second and the mass flow rate, respectively.

The `Multizone` package was implemented based on the multizone package described in [15], which has been contributed by the United Technologies Research Center (UTRC) for inclusion in the `Buildings` library. However, several changes have been done when migrating the models to Modelica 3.1, which led to a simpler implementation based on the stream function [9]. A comparison between the two implementations is described in Section 4.1.

3.2 Package Controls

The package `Controls` contains blocks that can be used in conjunction with the controls models from the Modelica Standard Library to implement controllers of building energy systems. The package `Controls.Continuous` has a new model `LimPID`, which can provide P, PI, PD, and PID controllers with limited output, anti-windup compensation and setpoint weighting. The package `Controls.SetPoints` has a new model `Table`, which allows setting a time-varying set point.

3.3 Package Fluid

The `Fluid` package contains component models for thermo-fluid flow systems. The level of modeling detail is comparable with the models of the `Modelica.Fluid` library. Most models in `Buildings.Fluid` extend models from `Modelica.Fluid` to form components that are typically needed when modeling building energy systems. The `Fluid` package is the largest package in the `Buildings` library, and it has 15 sub-packages. This section will discuss seven sub-packages to which new models have been added.

3.3.1 Package Fluid.Chillers

The package `Fluid.Chillers` contains two new chiller models. The first chiller model is an electric chiller based on the `EnergyPlus` chiller model `Chiller:Electric:EIR`. This model uses three functions to predict its capacity and its power consumption:

- a biquadratic function is used to predict its cooling capacity as a function of condenser entering and evaporator leaving fluid temperature,
- a quadratic function is used to predict its power input to cooling capacity ratio as a function of the part load ratio,
- a biquadratic function is used to predict its power input to cooling capacity ratio as a function of condenser entering and evaporator leaving fluid temperature.

The second implemented chiller model is an electric chiller based on the model by Hydeman et al. [10]. This model is also implemented in `EnergyPlus` as the model `Chiller:Electric:ReformulatedEIR` and is similar to the first chiller model. The main difference is that to compute its performance, this model uses the condenser leaving temperature instead of the entering temperature, and it uses a bicubic polynomial instead of a quadratic function to compute the part load performance. This model is reported to provide higher accuracy for variable-speed compressor drive and variable condenser water flow applications compared to the model `Chiller:Electric:EIR`.

The package `Fluid.Chillers.Data` contains performance data for more than 300 chillers.

3.3.2 Package Fluid.Interfaces

Similarly to `Modelica.Fluid.Interfaces`, there is a package `Buildings.Fluid.Interfaces`. It contains partial models for algebraic and dynamic components that exchange heat or

mass with one or two fluid streams. The model `PartialLumpedVolume` has been added to provide a base class for an ideally mixed fluid volume with the ability to store mass and energy. This model is similar to the partial model `PartialLumpedVolume` from `Modelica.Fluid.Interfaces`, except that it allows modeling the air humidity using a differential equation, while modeling the total mass balance using a steady-state equation.

3.3.3 Package Fluid.Actuators

The package `Fluid.Actuators` contains models of actuators. There are models of valves with two and three fluid ports and with various opening characteristics as well as models of air dampers. There are also models of motors that can be used in conjunction with the actuators.

The main change to this package was a redesign of the three-way-valves. The new implementation allows the optional addition of a fluid volume where the two fluid streams mix. The fluid volume can be conditionally added or removed based on the parameter `dynamicBalance`. The use of this fluid volume often leads to a more robust and faster simulation.

3.3.4 Package Fluid.HeatExchangers

This package contains algebraic and dynamic heat exchanger models, some of which compute condensation of water vapor that may occur at a cooling coil. Several new models have been added. For example, the model `HeatExchangers.DryEffectivenessNTU` describes a heat exchanger without water vapor condensation that is based on the effectiveness-NTU relation [11]. This model transfers heat in the amount of

$$\dot{Q} = \epsilon \dot{Q}_{max}, \quad (3)$$

where \dot{Q}_{max} is the maximum heat that can be transferred, and ϵ is the heat transfer effectiveness, defined as

$$\epsilon = f(NTU, Z, flowRegime), \quad (4)$$

where NTU is number of transfer units, Z is the ratio of minimum to maximum capacity flow rate and $flowRegime$ is the heat exchanger flow regime, such as parallel flow, cross flow or counter flow.

Also new in this package are the models `DryCoilCounterFlow` and `WetCoilCounterFlow`, which are finite volume models of counter flow heat exchanger without and with water vapor condensation if the air is cooled below its saturation temperature.

3.3.5 Package Fluid.Movers

This package contains component models for fans and pumps. Four new models have been added that can be parameterized by performance curves that compute pressure rise, electrical power draw or efficiency as a function of the flow rate. The four models differ in their implementation of the input signal, which can be a control signal, a prescribed speed, a prescribed mass flow rate or a prescribed pressure rise.

The models `FlowMachine_y` and `FlowMachine_Nrpm` take a control signal or a number of revolutions as an input, and then compute the resulting pressure difference for the current flow rate. The models `FlowMachine_dp` and `FlowMachine_m_flow` take the pressure difference or the mass flow rate as an input signal. The pressure difference or the mass flow rate will then be provided by the fan or the pump. These two models do not have a performance curve for the flow characteristics, because solving for the flow rate and the revolution at zero pressure difference can lead to a singularity.

All models can be configured to have a fluid volume at the low-pressure side. Adding such a volume sometimes helps the solver find a solution during initialization and time integration of large models.

All models compute the motor power draw P_{ele} , the hydraulic power input W_{hyd} , the flow work W_{flo} and the heat dissipated into the medium \dot{Q} . The governing equations are

$$W_{flo} = |\dot{V} \Delta p|, \quad (5)$$

$$W_{hyd} = W_{flo} + \dot{Q}, \quad (6)$$

$$\eta = W_{flo}/P_{ele} = \eta_{hyd} \eta_{mot}, \quad (7)$$

$$\eta_{hyd} = W_{flo}/W_{hyd}, \quad (8)$$

$$\eta_{mot} = W_{hyd}/P_{ele}, \quad (9)$$

where \dot{V} is the volume flow rate, Δp is the pressure rise, η is the overall efficiency, η_{hyd} is the hydraulic efficiency, and η_{mot} is the motor efficiency. All models take as a parameter an efficiency curve for the motor. This function has the form $\eta_{mot} = f(\dot{V}/\dot{V}_{max})$, where \dot{V}_{max} is the maximum flow rate. The models `FlowMachine_y` and `FlowMachine_Nrpm` set $\dot{V}_{max} = f_c(\Delta p = 0, r_N = 1)$ where $f_c(\cdot, \cdot)$ is a user-specified flow characteristic and r_N is the ratio of actual to nominal speed. Since `FlowMachine_dp` and `FlowMachine_m_flow` are not parametrized by the function $f_c(\cdot, \cdot)$, the parameter \dot{V}_{max} must be set by the user for these models.

These models have similar parameters as the models in the package `Modelica.Fluid.Machines`. However, the models in this package differ primarily in the following points:

- They use a different base class, which allows having zero mass flow rate for the fan and pump models. The models in `Modelica.Fluid.Machines` restrict the number of revolutions, and hence the flow rate, to be non-zero.
- For the model with prescribed pressure, the input signal is the pressure difference between two ports, and not the absolute pressure at the outlet port.
- The pressure calculations are based on total pressure in Pascals instead of the pump head in meters. This change has been made to avoid ambiguities in the parameterization if the models are used as a fan with air as the medium. The original formulation in `Modelica.Fluid.Machines` converts head to pressure using the density of the medium. For fans, head would be converted to pressure using the density of air. However, manufacturers of fans typically publish the head in millimeters water (mmH2O). Therefore, to avoid confusion when using these models with media other than water, our implementation uses total pressure in Pascals instead of head in meters.
- Additional performance curves have been added to the package `Buildings.Fluid.Movers.BaseClasses.Characteristics`.

3.3.6 Package Fluid.Sensors

This package consists of idealized sensor components that provide variables of the medium. These signals can be further processed with components of the `Modelica.Blocks` library. One can also build more realistic sensor models by further processing their output signal (e.g., by attaching the block `Modelica.Blocks.FirstOrder` to model the time constant of the sensor). The new version of the library increased the number of sensors from 4 to 22. Sensors are now available for density, enthalpy, mass flow rate, volume flow rate, pressure, relative humidity, dry bulb temperature, wet bulb temperature, species concentration, such as water vapor, and trace substances, such as carbon dioxide.

3.3.7 Package Fluid.Sources

This package contains models for fixed or prescribed boundary conditions for thermo-fluid systems. The new version of the library adds five different combinations of boundary sources. For example, the model `Boundary_ph` prescribes pressure, specific enthalpy, mass fraction and trace substances. The model `MassFlowSource.T` is an ideal flow source that produces a prescribed mass flow rate with prescribed temperature, mass fraction and trace substances.

3.3.8 Package Fluid.Storage

This package contains models of thermal energy storage tanks. For the model `StratifiedEnhanced`, a new model using the QUICK scheme [6] for the discretization of the fluid volume has been implemented. It computes a heat flux that needs to be added to each volume in order to give the results that a third-order upwind discretization scheme would give. If a standard third-order upwind discretization scheme were to be used, then the temperatures of the elements that face the tank inlet and outlet ports would overshoot by a few tenths of a Kelvin. To reduce this overshoot, the model uses a first order scheme at the boundary elements, and it adds a term that ensures that the energy balance is satisfied. Without this term, small numerical errors in the energy balance, introduced by the third order discretization scheme, would occur.

3.4 Package HeatTransfer

This package contains models for heat transfer elements. Based on a single-layer conduction model `ConductorSingleLayer`, the new version of the library adds the model `ConductorMultiLayer` for one-dimensional dynamic and steady-state heat conduction through multi-layer constructions. In addition, a convection model `Convection` and a model for combination of convection and conduction for opaque constructions `ConstructionOpaque` are also implemented. The models for heat convection are parameterized by different functions that compute heat convection based on temperature differences, or based on a constant value. These functions are available in the package `HeatTransfer.Functions.ConvectiveHeatFlux`.

3.5 Package Media

This package contains media models that can be used in addition to the models from `Modelica.Media`. Some of the media models

in this package are based on simplified state equations and property equations. The simplification can generally lead to a faster and more robust simulation compared to the models of `Modelica.Media`. The new version adds a sub-package `Media.GasesConstantDensity`. The models in this sub-package use a constant mass density to avoid having pressure as a state variable in mixing volumes. The advantage of using constant mass density is that fast transients introduced by a change in pressure are avoided. The disadvantage is that the dimensionality of the coupled nonlinear equation system is typically larger for flow networks.

3.6 Package Utilities

This package contains various utility models and functions, including diagnostics models, mathematical functions and a package for co-simulation with the Building Controls Virtual Test Bed (BCVTB). The BCVTB is a middle-ware that can connect Modelica with different simulation programs, such as MATLAB/Simulink, EnergyPlus and Radiance. The BCVTB can also link to building automation systems through a BACnet interface [2] and through analog/digital converters.

More interfaces, such as a model for the boundary condition for HVAC systems that use a medium with moist air `BCVTB.MoistAirInterface` and blocks for temperature conversions `BCVTB.to_degC`, `BCVTB.from_degC`, are introduced in the new version to facilitate the communication between Modelica and the BCVTB. For illustration, Fig. 2 shows an air-based heating system with an ideal heater and an ideal humidifier in the supply duct. The heating system is coupled to the BCVTB for co-simulation with EnergyPlus. The heater and humidifier are controlled with a feedback loop that tracks the room air temperature and room air humidity. These quantities are simulated in the EnergyPlus building simulation program. The component `bouBCVTB` models the boundary between the domain that models the air system (simulated in Modelica) and the room response (simulated in EnergyPlus).

To assess comfort conditions in rooms, the new version of the library adds a sub-package `Comfort`. This package contains a model that computes the thermal comfort according to the relations of Fanger [1]. In Fanger's model, the thermal sensation of a human is mainly related to the thermal

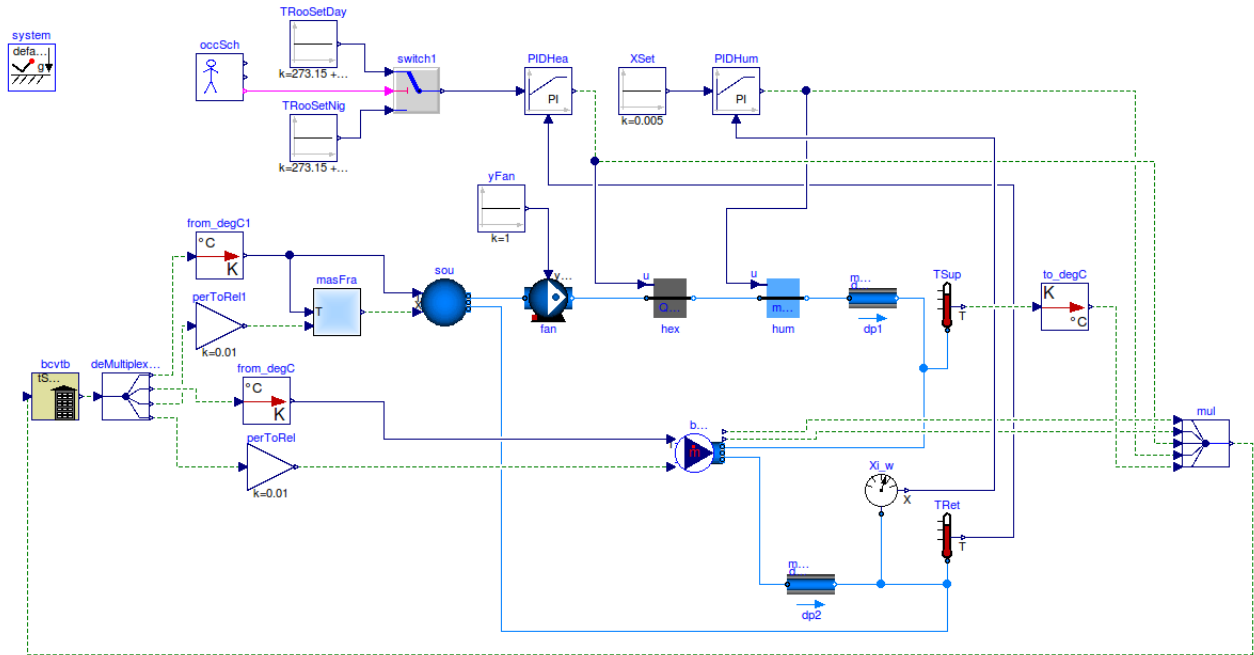


Figure 2: Modelica model that is used for co-simulation of a simple HVAC system that is connected to an EnergyPlus building model.

balance of its body. This balance is influenced by two groups of factors: personal and physical. The activity level and clothing thermal insulation of the subject form the group of personal factors, while the environmental parameters (e.g., air temperature, mean radiant temperature, air velocity, and air humidity) compose the group of physical factors. When the personal factors have been estimated and the physical factors have been measured, the average thermal sensation of a large group of people can be predicted by calculating the PMV (Predicted Mean Vote) index. The PPD (Predicted Percentage of Dissatisfied) index, obtained from the PMV index, provides information on thermal discomfort (thermal dissatisfaction) by predicting the percentage of people likely to feel too hot or too cold in the given thermal environment.

4 Applications

4.1 Multizone Air Flow Model

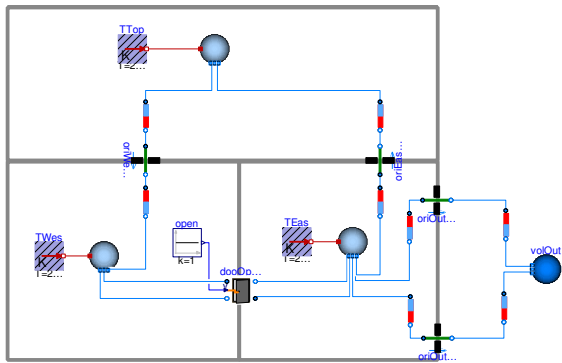
The multizone air flow model in the `Buildings` library is based on the library implemented by UTRC, which is presented in [15]. We converted the library to use the stream connectors [9], and implemented it in the `Buildings` library. Fig. 3 compares model diagrams for modeling airflow in three rooms. The diagram using stream connectors in Fig.3(a) is much simpler than the one with-

out the stream connectors in Fig. 3(b). The models became simpler, since with stream connectors, the `inStream` operator can be used to obtain the properties of the medium inside the volumes that are connected to the model that computes the stack effect.

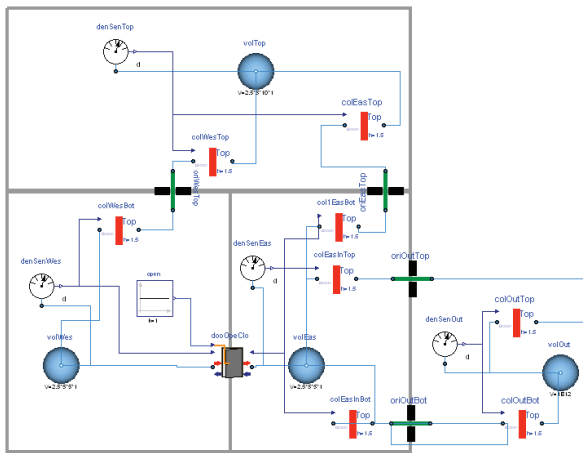
We also compared the mass flow rates through the door and the openings in the walls. The mass flow rates computed by the new version are the same as in the original implementation [15], which indicates that the implementation of the multi-zone airflow models using stream connectors was successful.

4.2 Modelica EnergyPlus Co-simulation for the Control of a VAV System

This example illustrates the use of co-simulation between Modelica and EnergyPlus through the BCVTB. In Modelica, we implemented a variable air volume flow system for a building with five thermal zones. The Modelica model implements the airflow network, the fans and heat exchangers, and the supervisory and local loop control. At the air inlet and outlet of the five thermal zones, an energy balance is made for the sensible and latent heat exchange. These heat flows are then added to the model of the thermal zones in EnergyPlus. EnergyPlus then computes the new room temperatures and water vapor concentrations, as



(a) With stream connectors.



(b) Without stream connectors.

Figure 3: Comparison between diagrams of three room problem implemented by Modelica multi-zone airflow models with and without stream connectors. Subfigure (b) is the model presented in [15].

well as the current weather conditions. EnergyPlus also computes the heat balance of the building and the lighting control as a function of the available daylight. Thus, the co-simulation allows users to utilize Modelica for the implementation and performance assessment of control sequences, and to utilize EnergyPlus for its whole building heat transfer and daylighting models.

Fig. 4 illustrates the HVAC system as implemented in Modelica. The total system model contained 970 components that led to a differential algebraic equation system with 5,000 scalar equations. The translated model had 90 continuous time states and 20 nonlinear systems of equations with dimensions up to 6. There are no numer-

ical Jacobians. The Modelica models are linked through the BCVTB to EnergyPlus, as shown in Fig. 5. For a more detailed description about this system and its simulation results, see [17].

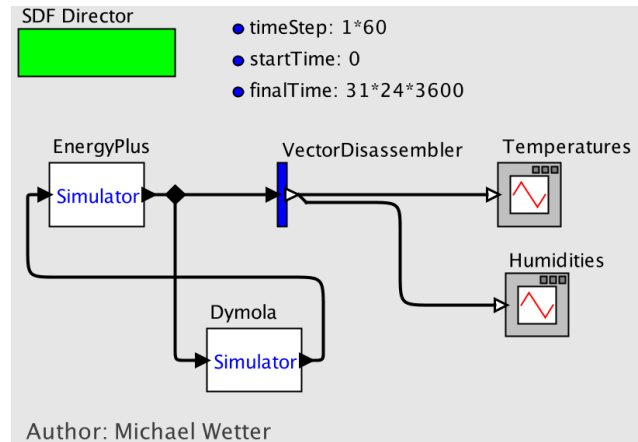


Figure 5: BCVTB model for the co-simulation between Modelica and EnergyPlus.

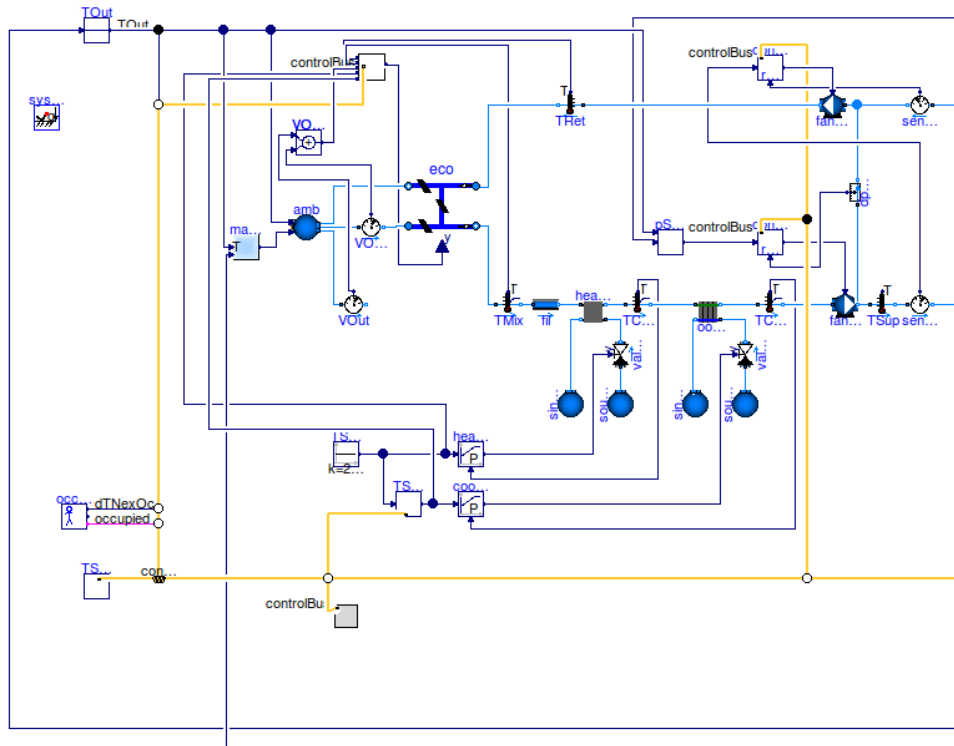
5 Ongoing Work

The next major addition to the Buildings library will be a package with models for a thermal zone that compute heat transfer through the building envelope and within a room. While the Buildings library can already be linked to EnergyPlus, for some situations, it is more practical to have all models implemented in Modelica. This requires the implementation of such a room model that can be used to assemble buildings with several thermal zones.

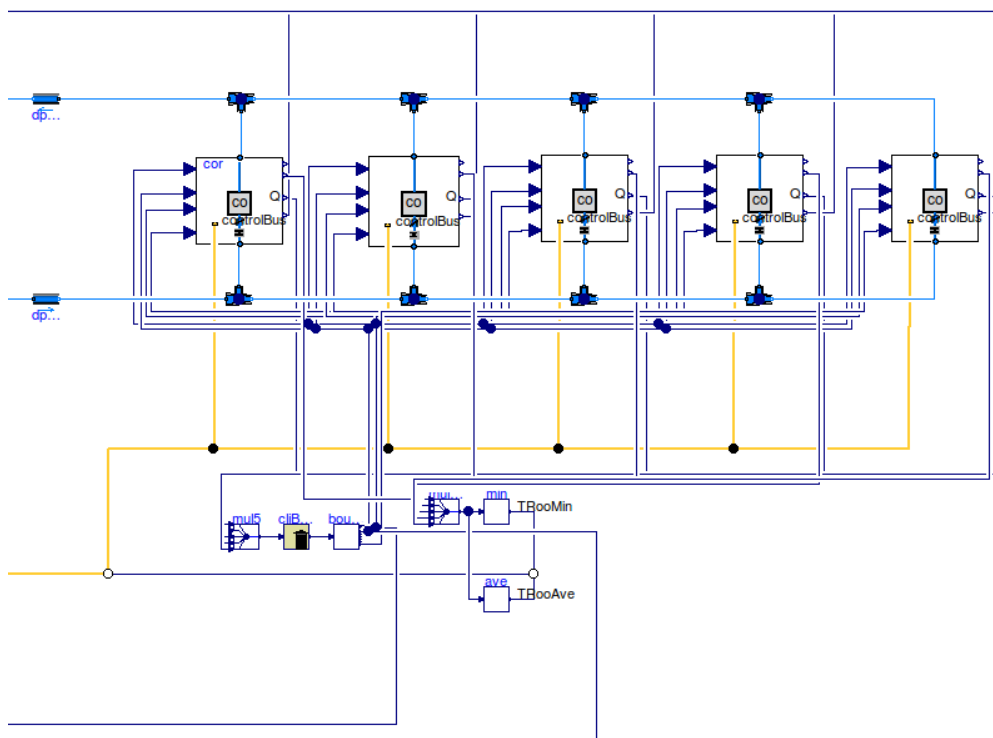
For the building envelope, we have implemented models for heat conduction through opaque multi-layer materials, which are available in the package Buildings.HeatTransfer. Currently, we are working on the implementation of models for window systems. The window model will compute a layer-by-layer radiation, convection and conduction heat balance, using equations that are similar to the ones in the Window 5 program [7].

We are also working on the implementation of models for short-wave, long-wave and convective heat transfer in a room. These models will be similar to the models described in [14].

To obtain boundary conditions, we have implemented a package Buildings.BoundaryConditions, which will be released in the next version. This package includes models for the sky black-body temperature, the path of the sun, and incidence angles on tilted surfaces. There will also be a weather



(a) Plant of VAV system.



(b) Distribution of VAV system with five thermal zones.

Figure 4: Modelica model of the VAV system that is linked to EnergyPlus for co-simulation.

data reader. Weather data can be obtained for free from <http://www.energyplus.gov>, and then converted to Modelica format with a Java program that is provided with the `Buildings` library.

6 Conclusion

The Modelica `Buildings` library has been significantly expanded since the last Modelica conference. A new `Airflow` package has been added for indoor air flow simulation. Many models have been added into existing packages to provide more functionality, and existing models have been revised to improve the numerical efficiency. However, for large fluid flow systems with feedback control, such as the ones shown in Fig. 4, computing consistent initial conditions and performing the time integration can still lead to numerical problems. The robust simulation of such systems still requires further research.

7 Acknowledgments

This research was supported by the Assistant Secretary for Energy Efficiency and Renewable Energy, Office of Building Technologies of the U.S. Department of Energy, under Contract No. DE-AC02-05CH11231.

We would also like to thank the United Technologies Research Center for contributing the `Multizone` package to the `Buildings` library.

References

- [1] ASHRAE. ASHRAE fundamentals, 1997.
- [2] ASHRAE. ANSI/ASHRAE Standard 135-2004, BACnet, a data communication protocol for building automation and control networks, 2004.
- [3] J. Axley. Multizone airflow modeling in buildings: History and theory. *HVAC&R Research*, 13(6):907–928, 2007.
- [4] DOE. Buildings energy data book. Technical Report DOE/EE-0325, Department of Energy, 2009.
- [5] W. S. Dols and G. N. Walton. CONTAMW 2.0 user manual, multizone airflow and contaminant transport analysis software. Technical Report NISTIR 6921, National Institute of Standards and Technology, 2002.
- [6] J. H. Ferziger and M. Peric. *Computational methods for fluid dynamics*. Springer, Berlin, New York, 3rd, rev. edition, 2002.
- [7] E. U. Finlayson, D. K. Arasteh, C. Huizenga, M. D. Rubin, and M. S. Reilly. *WINDOW 4.0: Documentation of Calculation Procedures*. Lawrence Berkeley National Laboratory, Berkeley, CA, USA, 1993.
- [8] R. Franke, F. Casella, M. Otter, K. Proelss, M. Sielemann, and M. Wetter. Standardization of thermo-fluid modeling in `modelica.fluid`. In F. Casella, editor, *Proc. of the 7-th International Modelica Conference*, Como, Italy, Sept. 2009.
- [9] R. Franke, F. Casella, M. Otter, M. Sielemann, H. Elmqvist, S. E. Mattsson, and H. Olsson. Stream connectors: an extension of `modelica` for device-oriented modeling of convective transport phenomena. In F. Casella, editor, *the 7th International Modelica Conference*, Como, Italy, 2009.
- [10] M. Hydeman, N. Webb, P. Sreedharan, and S. Blanc. Development and testing of a reformulated regression-based electric chiller model. *ASHRAE Transactions*, 108(2), 2002.
- [11] F. P. Incropera and D. P. D. Witt. *Fundamentals of Heat and Mass Transfer*. John Wiley & Sons, 5th edition, 2001.
- [12] A. C. Megri and F. Haghghat. Zonal modeling for simulating indoor environment of buildings: Review, recent developments, and applications. *HVAC&R Research*, 13(6):887–905, 2007.
- [13] P. V. Nielsen. Computational fluid dynamics and room air movement. *Indoor Air*, 14:134–143, 2004.
- [14] M. Wetter. *Simulation-Based Building Energy Optimization*. PhD thesis, University of California, Berkeley, 2004.
- [15] M. Wetter. Multizone airflow model in `modelica`. In *the 5th International Modelica Conference*, pages 431–440, Vienna, Austria, 2006.
- [16] M. Wetter. Modelica library for building heating, ventilation and air-conditioning systems. In *the 7th International Modelica Conference*, Como, Italy, 2009.
- [17] M. Wetter. Co-simulation of building energy and control systems with the building controls virtual test bed. *In press: Journal of Building Performance Simulation*, 2010.
- [18] W. Zuo and Q. Chen. Real-time or faster-than-real-time simulation of airflow in buildings. *Indoor Air*, 19(1):33–44, 2009.

Object-oriented sub-zonal room models for energy-related building simulation

Marco Bonvini*, Alberto Leva

Dipartimento di Elettronica e Informazione, Politecnico di Milano
Via Ponzio 34/5, 20133 Milano, Italy
{bonvini,leva}@elet.polimi.it

*PhD student at the Dipartimento di Elettronica e Informazione

Abstract

Simulation is important to evaluate the energy-related performance of a building, and for reliable results, reproducing the behaviour of the contained air volumes is particularly relevant. For such a purpose, fully mixed models (i.e., for instance, a single temperature per room) easily prove inadequate, while Computational Fluid Dynamics (CFD) ones are too complex, and difficult to formulate in a modular manner, to the detriment of their usefulness if the simulator has to be used throughout the project, and not only to assess its final result. This manuscript presents an intermediate solution based on the Modelica language.

Keywords: Building simulation; energy optimisation; object-oriented modelling; modular modelling; scalable detail.

1 Introduction

In the research on building simulation, probably the toughest challenge is to deliver tools that can effectively confront the multi-physic nature of such complex systems. The energy performance of a building in fact results from phenomena of heterogeneous type (hydraulic, thermal, electric and so forth) together with the operation of several control systems and the actions of the inhabitants. Better still, energy performance is determined by the *interaction* of all those phenomena [22].

Traditionally, the design of a building is treated in practice as the partially disjoint (explanations follow) design of its “subsystems”. Although there is no standardised nomenclature, in fact, virtually the totality of engineering tools broadly distinguish (a) the “building” *stricto sensu*, i.e., walls, doors, windows and so on, (b) the contained air volumes, possibly divided in zones, (c) the Heating, Ventilation and Air Condition-

ing (HVAC) system, (d) automation and control systems, and (e) energy sources/sinks owing to the building utilisation, e.g., the heat released by occupants, industrial machines, or whatever is installed. The subsystems’ interaction is accounted for by having some of them provide boundary conditions for the design of some other.

This is apparently very far from a really integrated approach, whence the term “partially disjoint” applied above to current design practices, but tools that address the simulation of all (or at least part) of the subsystems in a coordinated way are at present little more than research objects [12, 23, 22].

The main reason for such a *scenario* are the very different issues posed by the various subsystems. For example, control system models are made of oriented blocks and may need sometimes a continuous-time and sometimes a digital representation depending on the simulation purpose; models for HVAC, conversely, live invariantly in the continuous-time domain, but are typically zero- or one-dimensional, while models of phenomena that occur in *continua* such as a wall or an air volume often cannot avoid three-dimensional spatial distributions. As a result, it is difficult to devise simulation models that address all the necessary phenomena, and can be organised in a modular way, to the advantage of their construction, parametrisation, and maintenance.

2 Literature review

In building simulation, modelling air volumes requires to treat temperature and heat flow distributions in a coordinated way with respect to how the same distributions are addressed in solid bodies (e.g., walls) and possibly other fluids (e.g., heat conveying ones).

A widely used modelling paradigm is that of *zonal*

models or *zoning*, where the air within a building is split into zones, typically rooms. Zones are *macro*-volumes with respect to the scale of the spatial temperature and flow distribution, allowing for a small number of simulated variables, but posing non-trivial problems for the determination of average fluid properties.

However the zonal approach allows to clearly characterise the relationships between a zone and the adjacent entities, thus to create modular models, typically distinguishing “storage” elements (like the air volumes) and “flow” ones, that describe the mass and energy flow among the storages. Many literature works and engineering tools adopt the zonal approach: examples are COMIS, CONTAM, POMA, see [4, 7, 20, 10], and EnergyPlus [3].

On the opposite side with respect to the zonal paradigm stands the CFD one, that provides far more accuracy, but the computation-intensive, and does not allow to separate easily the (partial) differential equations that hold within a volume from the boundary conditions, making the creation of modular models a complex task. There exist CFD tools applied to buildings, e.g. Fluent [8, 9], but their use is most frequently limited to static problems, and hardly ever considered in system level studies.

In recent years, various attempts are being made to join air models with the description of other elements such as containment, HVAC, and possibly the electric system, the behaviour of inhabitants, weather conditions, and so forth, see e.g. [5]. To achieve such ambitious a goal, a promising paradigm is Object-Oriented Modelling (OOM), see [21], and in particular the Modelica language [17] and [6]. To date, however, OOM-related research enforces modularity by relying on the zonal models idea, which is the easiest way to go, but definitely not the most accurate.

In the last years a somehow intermediate proposal, termed *sub-zonal modelling* or *sub-zoning*, was formulated in an attempt to join the best of zoning and CFD [16, 14, 25]. This improves accuracy at the cost of a (moderate) complexity increase, but still poses non-trivial issues with respect to modularity, especially if air models need to be connected to heterogeneous entities such as prescribed boundary conditions (e.g.. from the external environment), walls, piping, and so on.

This manuscript aims at filling the gap just sketched, proposing the innovative model structuring described below, and maintaining compatibility with other Modelica-related research on the matter [22, 23].

3 The proposed modelling approach

With respect to the way equations are formulated, the distinctive characteristic of this work is that the momentum balance is introduced *explicitly*, contrary to previous (specifically, non-CFD) literature. An *ad hoc* spatial discretisation of said equation makes it natural to account for gravity and any possible other motion driving force. With respect to model structuring and implementation, the object-oriented paradigm is strictly followed.

3.1 Balance equations

This work starts from the three Navier-Stokes equations for mass, energy and momentum, that for the purpose of this work can be written as

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{v}) = 0 \quad (1a)$$

$$\frac{\partial (\rho e)}{\partial t} + \nabla \cdot (\rho \mathbf{v} c_p T) = \nabla \cdot (k \nabla T) \quad (1b)$$

$$\frac{\partial (\rho \mathbf{v})}{\partial t} + \nabla \cdot (\rho \mathbf{v} \mathbf{v}) + \nabla P = \nabla \mathbf{f} \quad (1c)$$

where the scalars p , T , e and ρ are respectively the fluid pressure, temperature, specific energy and density, the vectors \mathbf{v} and \mathbf{f} are the fluid velocity and the possible motion driving forces, and the scalar parameters k and c_p are the fluid thermal conductivity and constant-pressure specific heat capacity. In the cases of interest for this research the fluid (air) can be considered a mixture of ideal gases, which allows to express the specific energy e as $c_v T$, where c_v is the constant-volume specific heat capacity. As further simplification, Newtonian fluid model is adopted, thereby rewriting (1c) as

$$\frac{\partial (\rho \mathbf{v})}{\partial t} + \nabla \cdot (\rho \mathbf{v} \mathbf{v}) + \nabla P = \nabla \cdot (\mu \nabla \mathbf{v}) \quad (2)$$

where μ is the fluid dynamic viscosity, and then the scalar projection is brought in.

The set of equations (1a), (1b) and (2), are spatially discretised with reference to finite volume elements (not necessary uniform) of parallelepiped shape. To deal with said spatial discretisation, a *staggered* grid of points [15, 19] is defined in the spatial domain of interest, as illustrated in figure 1. For simplicity the figure refers to a 2-dimensional case, extension to 3-dimensional space is straightforward. In the

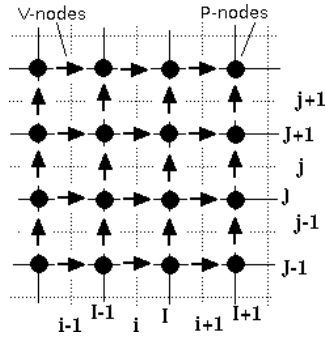


Figure 1: Staggered grid adopted for the discretisation (2D case).

grid there are elements that represent the volumes (circles) while other elements (horizontal and vertical arrows) that represent the coupling element between adjacent volumes, or between volumes and boundaries. Within these elements the balance equations of mass, energy and momentum are discretised. In particular the mass and energy ones are integrated within the volumes while the momentum balance ones are within the coupling elements.

3.2 Other equations

To complete the model it suffices to complement the balance equations introduced and discretised so far with those pertaining to the fluid state, the energy transfers not associated to fluid motion, and possibly the required turbulence model.

3.2.1 Fluid state

The fluid considered here is air, treated as a mixture of ideal gases: 78 % of nitrum and 22 % of oxygen. Instead of using the ideal gas relationship, in order to simplify the model, the linearisation

$$\rho = \frac{p}{R^*T} \quad (\text{ideal gas}) \quad (3a)$$

$$\rho = \rho_o + \frac{1}{R^*T_o}P - \frac{p_o}{R^*T_o^2}(T - T_o) \quad (\text{linearised}) \quad (3b)$$

is here used, where ρ is the fluid density, R^* is the specific ideal gas constant, T the absolute temperature of the gas, p the absolute pressure, ρ_o gas density at the linearisation point, p_o and T_o are the values of absolute pressure and temperature at the same point. Notice the use of the *relative* pressure $P = p - p_o$, in order to

avoid numerical errors due to the large absolute pressure values. The discrepancy between the ideal and the linearised model is very limited in the typical operating range. In addition to the state equation, also the specific energy and enthalpy equations are necessary: here they are simply written, as partially anticipated, in the form

$$e = c_v T \quad (\text{specific energy}) \quad (4a)$$

$$h = e + \frac{P}{\rho} \quad (\text{specific enthalpy}) \quad (4b)$$

3.2.2 Thermal exchanges not associated with fluid motion

The heat fluxes due to thermal air conduction are computed with the Fourier-like law

$$Q_{A \rightarrow B} = \frac{\gamma \cdot A_{AB}}{d_{AB}} \cdot (T_A - T_B) \quad (5)$$

where $Q_{A \rightarrow B}$ is the thermal power flowing from volume A to volume B, γ is the fluid's thermal conductivity (for air $\gamma = 0.026 [W/mK]$), A_{AB} is the surface shared by the adjacent volumes, d_{AB} is the distance between the volume centres, and $T_{A,B}$ are respectively the temperatures of volumes A and B. Notice that (5) can be shown to be the discretisation of the right hand side of (1b).

In addition, when dealing with boundary conditions such as walls, there is a convective heat transfer instead of a conductive one. The thermal power flowing from to an adjacent volume can thus be calculated as

$$Q_{Wall \rightarrow Volume} = h \cdot A \cdot (T_{Wall} - T_{Volume}) \quad (6)$$

where h is the convective heat transfer, A is the portion of area shared by the volume and the wall, T_{Wall} and T_{Volume} are respectively the temperature of the wall and of the volume.

3.2.3 Simple turbulence modelling

For laminar flows, the results provided are natively accurate and reliable. As witnessed by the CFD literature, the same is not true for turbulent flows. The introduction of a *turbulence model* is most common way to solve this problem, and a lot of such have been studied and implemented.

The solution used here is based on the idea of “zero-equation” turbulence modelling, first introduced by Prandtl, at the beginning of the nineteenth century.

After Prandtl's work, many effort were made to extend the applicability of his theory [24], and among the so obtained results, those of [2] were chosen for this work, given their simplicity and the available validations in a context (HVAC) similar to that addressed here. Starting from the quoted work, the viscosity μ in the momentum equation, when dealing with turbulent flows, is thus replaced by the "effective" dynamic viscosity

$$\mu_{eff} = \mu + \mu_T \quad (7)$$

that is a sum of the intrinsic fluid dynamic viscosity μ and a turbulent viscosity μ_T , that according to [2] comes from an algebraic function of local mean velocity V and at a length scale l given by

$$\mu_T = 0.03874\rho V l \quad (8)$$

The function (8) was here implemented considering as mean velocity V the velocity of the air flowing through the coupling element, and as length scale l as the distance between the centres of the volumes linked by the coupling element.

4 Discretised equations

In order to develop a model representing the air contained within a room (or more in general an ambient) the basic equations of mass (1a), energy (1b) and momentum (2) preservation have to be discretised as anticipated in section 3. These equations are discretised accordingly to the grid structure shown in figure (1) and finite volume elements (not necessary uniform) of parallelepiped shape, $A_{x,y,z}$ being the areas of the two element faces having as normal the x , y and z axis versors, and V the element volume. The mass and energy equations are treated in quite standard a manner, giving rise to the two scalar ones

$$V \frac{\partial \rho}{\partial t} = w_{x^-} + w_{x^+} + w_{y^-} + w_{y^+} + w_{z^-} + w_{z^+} \quad (9a)$$

$$\begin{aligned} V \frac{\partial(\rho e)}{\partial t} = & w_{x^-} \cdot h_{x^-} + w_{x^+} \cdot h_{x^+} + w_{y^-} \cdot h_{y^-} \\ & + w_{y^+} \cdot h_{y^+} + w_{z^-} \cdot h_{z^-} + w_{z^+} \cdot h_{z^+} \\ & + Q_{x^-} + Q_{x^+} + Q_{y^-} \\ & + Q_{y^+} + Q_{z^-} + Q_{z^+} + Q_g \end{aligned} \quad (9b)$$

where w_{a^-,a^+} are the mass flow rates across the two surfaces orthogonal to axis a , assumed positive when entering the element, h_{a^-,a^+} are the specific enthalpies transported by fluid motion across said surfaces, Q_{a^-,a^+} the thermal powers crossing the same

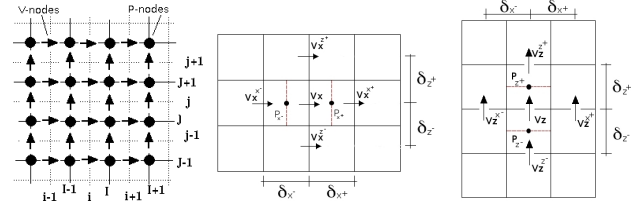


Figure 2: Staggered grid evidencing pressure and velocities nodes (left), and grid application to the x (centre) and z (right) momentum equations.

surfaces without fluid motion (due e.g. to diffusion), and Q_g the thermal power possibly generated within the volume.

As anticipated before, contrary to previous works that introduced empirical correlation instead of introducing the momentum balance equation, here the problem is treated. In this work is adopted an *ad hoc* approximation for the velocities' second derivatives, and a corresponding treatment of the boundary volume elements.

First the Newtonian fluid simplification is adopted, thereby rewriting (1c) as (2). Second the convective term $\nabla \cdot (\rho \mathbf{v} \mathbf{v})$ has been intentionally neglected since it is not relevant in the context addressed. Thus after the mentioned manipulation, and considering for brevity the 2D case

$$\begin{aligned} \frac{\partial \rho v_x}{\partial t} = & + \frac{\partial p}{\partial x} \\ & + \frac{\partial}{\partial x} \left(\mu \frac{\partial v_x}{\partial x} \right) + \frac{\partial}{\partial z} \left(\mu \frac{\partial v_x}{\partial z} \right) \end{aligned} \quad (10a)$$

$$\begin{aligned} \frac{\partial \rho v_z}{\partial t} = & - \rho g + \frac{\partial p}{\partial z} \\ & + \frac{\partial}{\partial x} \left(\mu \frac{\partial v_z}{\partial x} \right) + \frac{\partial}{\partial z} \left(\mu \frac{\partial v_z}{\partial z} \right) \end{aligned} \quad (10b)$$

Spatial discretisation is managed as described in the following, referring to the two-dimensional case with the x and z (vertical) axes only (the three-dimensional extension is trivial and would only unnecessarily complicate the notation). With reference to the staggered grid (figure 1), these equations are discretised within volumes that are centred respectively on horizontal and vertical arrows.

For the discretisation of the x and z -axis momentum equations (10) the grid of figure 2 (centre and right) is considered, and in this treatise the only moving force introduced is gravity (directed as the negative z axis); generalisations to other forces are straightforward. In figure 2 and the following analogous ones, arrows indicate the positive velocity direction assumed when discretising the momentum equations.

First, consider the x equation. The velocity to be computed is V_x , while V_x^{x-} , V_x^{x+} , V_x^{z-} and V_x^{z+} denote respectively the x component of velocities in the “west”, “east”, “south” and “north” surrounding velocity nodes—a notation that figure 2 (centre) should make self-explanatory. Analogously, p_{x-} and p_{x+} are the pressures of the west and east pressure nodes. The distances between the node in which the V_x velocity is computed and the surrounding ones where the x velocity components are accounted for, are denoted by δ_{x-} , δ_{x+} , δ_{z-} and δ_{z+} .

In (10a), three terms have to be spatially discretised. The first one is $\partial p/\partial x$, that simply yields

$$\frac{\partial p}{\partial x} \approx \frac{p_{x+} - p_{x-}}{\frac{1}{2}\delta_{x-} + \frac{1}{2}\delta_{x+}} \quad (11)$$

The second term is $\partial(\mu\partial v_x/\partial x)/\partial x$. Assuming the viscosity μ uniform in the volume element, one can write

$$\mu \frac{\partial}{\partial x} \left(\frac{\partial v_x}{\partial x} \right) = \mu \frac{\partial^2 v_x}{\partial x^2} \quad (12)$$

For the partial second derivative of the x velocity with respect to x , a second order polynomial function $V_x \approx ax^2 + bx + c$ is taken as local approximant, consistently with the quasi-3D spatial discretisation of a second derivative, and readily parametrised as

$$V_x^{x-} = a(x - \delta_{x-})^2 + b(x - \delta_{x-}) + c \quad (13a)$$

$$V_x = ax^2 + bx + c \quad (13b)$$

$$V_x^{x+} = a(x + \delta_{x+})^2 + b(x + \delta_{x+}) + c \quad (13c)$$

The required second derivative approximation is thus $2a$, which yields

$$a = \frac{\frac{V_x^{x+} - V_x}{\delta_{x+}} - \frac{V_x - V_x^{x-}}{\delta_{x-}}}{\delta_{x-} + \delta_{x+}} \quad (14)$$

allowing to reformulate (12) as

$$\mu \frac{\partial^2 v_x}{\partial x^2} \approx 2\mu \frac{\frac{V_x^{x+} - V_x}{\delta_{x+}} - \frac{V_x - V_x^{x-}}{\delta_{x-}}}{\delta_{x-} + \delta_{x+}} \quad (15)$$

The last term to be discretised is $\partial(\mu\partial v_x/\partial z)/\partial z$. Also in this case since the viscosity is assumed uniform in the element, yielding

$$\mu \frac{\partial}{\partial z} \left(\frac{\partial v_x}{\partial z} \right) = \mu \frac{\partial^2 v_x}{\partial z^2} \quad (16)$$

and for the second partial derivative of the x velocity with respect to z as a second order polynomial approximant is again taken. With a reasoning similar to that

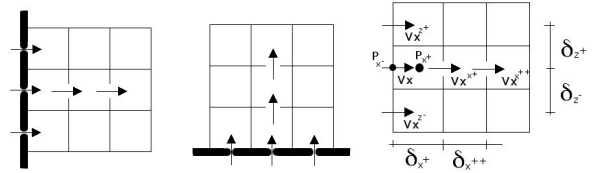


Figure 3: Boundary layer for x (left) and z (centre) velocity, and grid for the x momentum equation on the west boundary (right).

previously reported, (16) is thus approximated as

$$\mu \frac{\partial^2 v_x}{\partial z^2} \approx 2\mu \frac{\frac{V_x^{z+} - V_x}{\delta_{z+}} - \frac{V_x - V_x^{z-}}{\delta_{z-}}}{\delta_{z-} + \delta_{z+}} \quad (17)$$

Considering the z momentum equation, the same approach can be followed. The only difference with respect to the x axis is the presence of gravity, that does not need any discretisation.

The discretised momentum equations reported so far are valid in the volumes within a cavity (a room, a duct, a box...) but apparently not for the volumes at the cavity boundaries. As shown in figure 3 (left and centre), velocity nodes referring to volumes at the boundary may not have west/east neighbours for the x velocity case, and may not have north/south neighbours for the z case. A special momentum equation discretisation is thus required for boundary velocities.

For brevity, consider the x case in a velocity node located on the west cavity boundary (the other cases are analogous) illustrated in figure 3 (right). The equation to be discretised is (10a), requiring the pressure gradient along x and the second derivatives of the x velocity with respect to x and z . The pressure gradient can be discretised as

$$\frac{\partial p}{\partial x} \approx \frac{p_{x+} - p_{x-}}{\delta_{x+}} \quad (18)$$

For the second partial derivative with respect to z , (17) is still valid. The only change is in the second derivative of the x velocity with respect to x . A second order polynomial function can still be used, but this time the system to solve in order to parametrise it is

$$V_x = ax^2 + bx + c \quad (19a)$$

$$V_x^{x+} = a(x + \delta_{x+})^2 + b(x + \delta_{x+}) + c \quad (19b)$$

$$V_x^{x++} = a(x + \delta_{x+} + \delta_{x++})^2 + b(x + \delta_{x+} + \delta_{x++}) + c \quad (19c)$$

and thus to

$$\mu \frac{\partial^2 v_x}{\partial x^2} \approx 2\mu \frac{\frac{V_x^{x++} - V_x^{x+}}{\delta_{x++}} - \frac{V_x^{x+} - V_x}{\delta_{x+}}}{\delta_{x+} + \delta_{x++}} \quad (20)$$

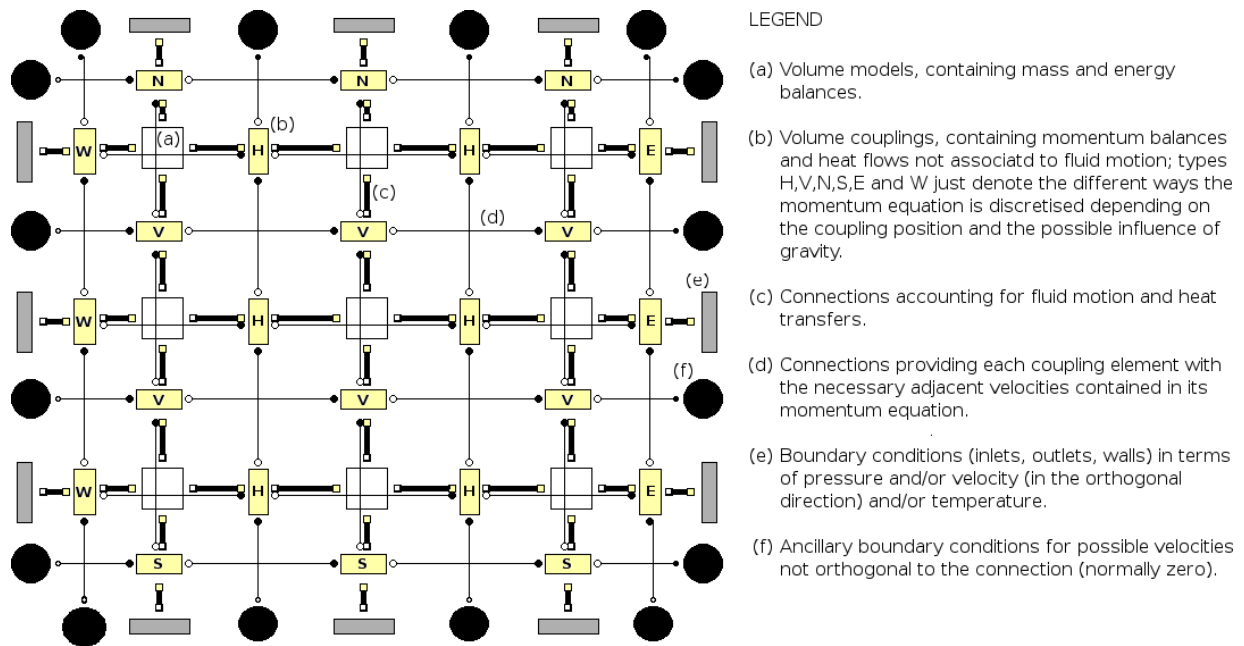


Figure 4: Modelica connection scheme for a $3 \times 3(\times 1)$ room.

5 Modelica implementation

The adopted discretisation approach is a very easy modularisation of the obtained models. To show that, based on the considerations above, some words are now spent (exhausting the matter is not possible for space limitations) how the devised models are realised in the Modelica language [13, 18].

The grid on which the Navier-Stokes equation are discretised can be represented in a modular way, where volume models are connected together with coupling models. Volume models are of a single type, while coupling ones can be of “internal” or “boundary” type. The staggered grid thus corresponds in Modelica to a modular structure composed only by the main model classes *volume*, *coupling* and *boundary*, with a uniform interface.

- *Volume models* contain the mass balance, the energy balance, the fluid state equation, the specific energy equation, and the specific enthalpy equation.
- *Coupling models* contain the momentum balance, the turbulence model, and the heat flow equation.
- *Boundary models* are similar to coupling ones but also contain the heat equation.
- *Connectors* are in fact very simple with the adopted choices, and are of two types. A first type contains the information on the fluid state

and that used by the coupling elements to solve the momentum equation, namely relative pressure, absolute temperature, fluid velocity through the face, heat flow rate through the face, specific enthalpy flowing through the face, density of the fluid flowing through the face, velocities associated the other faces of the volume, and sizes of the volume. The second type connects coupling/boundary elements providing the velocity of surrounding coupling elements, and the distance between said elements.

As a result, constructing a compound model is very easy by means of array structures. A compound model is spatially parametrised by just providing its dimensions, and the number of volume divisions (not necessarily evenly spaced) along the coordinate axes. At present only (compounds) of parallelepiped shape elements are allowed, extensions will be introduced in the future.

Also, replacing the fluid state equation with a different one is very straightforward, as is modifying the turbulence model. For example, figure 4 shows how a room model with $3 \times 3(\times 1)$ sub-zonal volumes is viewed in a Modelica graphical editor. Note that in said figure and in the following analogous ones relative to the examples, 2D arrangements are used for simplicity and/or consistence with the literature references used for the validation, but of course the devised formalism is natively quasi-3D.

6 Modelica Models

This section describes the Modelica implementation of the most important components that, once connected together, compose the model of the air within a room.

6.1 Connectors

The 2-dimensional room model in figure 4 shows how the elements (volumes, coupling elements and boundary conditions) are connected. Said elements are strongly interacting, since each coupling element has to know at least three velocity values (coming from of its neighbours) in order to compute the momentum balance (see section 4). To fulfil this need, specific connectors are defined, that convey information about velocities (and other useful data such as geometrical ones). The requirements expressed so far have lead to the implementation of two kinds of connectors: a first one to connect volumes and layers, that contains physical and velocities/geometrical information, and a second one to connect layers with layers, that contains velocities/geometrical informations only. The Physical/Information connector is written in Modelica as follows.

```
connector faceA
  SI.Pressure P;
  SI.Temperature T;
  SI.velocity v;
  flow SI.HeatFlowRate q;
  input Real h_ba; // Note: there exists also a B-type
  output Real h_ab; // connector, where input and output
  input Real rho_ba; // are reversed: an A-type is always
  output Real rho_ab; // connected to a B-type
  input Real V_ba;
  output Real V_ab;
  ... other velocities ...
  input Real dx_ba;
  output Real dx_ab;
  ... other distances ...
end faceA;
```

The Information connector, conversely, corresponds to the following Modelica code.

```
connector VelocityA
  input Real V_ba; // Note: here too a B-type exists,
  output Real V_ab; // see above
  input Real d_ba;
  output Real d_ab;
end VelocityA;
```

6.2 Volume

The volume represents a portion of the air where temperature and pressure are assumed as uniform. It contains the mass and energy balances, and corresponds to the following Modelica code (only the essential parts are reported).

```
model Volume
  SI.Density rho "air density";
  SI.Mass m "mass of air contained within the volume";
  SI.Pressure P "relative pressure of the air within the volume";
  SI.Temperature T(start = Tstart) "temp. of the air within the volume";
```

```
SI.SpecificEnthalpy h "spec. enthalpy of the air within the volume";
SI.SpecificEnergy e "spec. energy of the air within the volume";
...
other variables omitted for brevity
...
parameter SI.Temperature Tstart = 273.15 + 20 "initial temperature";
parameter SI.Pressure Po = 101325 "Pressure lin. value";
parameter SI.Temperature To = 298.15 "Temperature lin. value";
parameter SI.Density rho_o = Po/(R*To) "Density initial value";
parameter SI.SpecificHeatCapacity R=CONST.R*1000/28.97;
parameter SI.SpecificHeatCapacity cv = 1006 "air spec. heat";
parameter SI.Length height = 0 "distance between the volume ceiling";
parameter SI.Acceleration g = CONST.g_n "constant gravity acceleration";
parameter Real ComprCoeff = 1/(R*To) "air compressibility coefficient";
parameter Real ThermalExpCoeff = Po/(R*To^2) "air thermal exp. coeff.";
...
geometrical parameters
...
// thermal connector
Modelica.Thermal.HeatTransfer.Interfaces.HeatPort_a HeatPort;
// Volumes connectors
faceA W "connector of west face";
faceB E "connector of east face";
faceA S "connector of south face";
faceB N "connector of north face";
faceA BO "connector of bottom face";
faceB TO "connector of top face";
initial equation
  rho = rho_o + ComprCoeff*(rho*g*height) - ThermalExpCoeff*(Tstart-To);
  P = rho*g*height;
equation
  // The linearised PV=nRT gas relationship
  rho = rho_o + ComprCoeff*P - ThermalExpCoeff*(T-To);

  // mass of the air volume
  m = rho*V;

  // mass conservation
  der(m) = ww + we + wbo + wto + ws + wn;

  // incoming and outgoing air mass flows
  // along (x,y,z)-directions
  ww = dy*dz*W.v*(if noEvent(W.v>0) then W.rho_ba else rho);
  ...
  wn = - dy*dx*N.v*(if noEvent(N.v<0) then N.rho_ab else rho);

  // specific energy
  e = cv*T;

  // specific enthalpy
  h = e + P/rho;

  // energy balance
  der(m*e) = W.q + E.q + N.q + S.q + BO.q + TO.q + h_flow
            + HeatPort.Q_flow;

  // enthalpy flows
  h_flow = + ww*(if noEvent(ww>0) then W.h_ba else h)
            ...
            + wn*(if noEvent(wn>0) then N.h_ab else h);
  ...
  other equations omitted for brevity
  ...
end Volume;
```

6.3 Coupling element

The coupling element represents the interaction between two adjacent volumes (in the Modelica code, the coupling element model is called layer, boundary layer when it is on the boundary of the domain). It contains the momentum balance and heat transfer equations. There are various coupling elements, each one intended for computing the momentum balance along a given direction (x, y or z) and in a particular position (at the room boundary or not). The code reported below refers to a coupling element that computes the momentum balance equation along the z-direction, not at the room boundary (here too only the most important parts are shown).

```
model LayerZ
  SI.Velocity Vz "velocity of the air";
  parameter Real gamma=0.026 "air thermal conductivity";
```

```

parameter Real mu = 1.83e-5 "air viscosity, laminar framework";
parameter SI.Acceleration g = CONST.g_n
    "standard acceleration gravity on earth";
...
other geometrical parameters
...
// connectors
faceA B0 "bottom connector: Volume-Layer";
faceB T0 "top connector: Volume-Layer";
VelocityA W "west connector: Layer-Layer";
VelocityB E "east connector: Layer-Layer";
VelocityA S "south connector: Layer-Layer";
VelocityB N "north connector: Layer-Layer";
initial equation
// initialization for the considered average density
rho = 0.5*B0.rho_ba + 0.5*T0.rho_ab;
equation
// average and smoothed density
rho + 1*der(rho) = 0.5*B0.rho_ba + 0.5*T0.rho_ab;

// momentum balance equation
rho*der(Vz) = F
+2*MU*((W.V_ba - Vz)/dx_w - (Vz - E.V_ab) /dx_e)
/(dx_w+dx_e)
+2*MU*((B0.V_ba - Vz)/dz_bo - (Vz - T0.V_ab)/dz_to)
/(dz_bo+dz_to)
+2*MU*((S.V_ba - Vz)/dy_s - (Vz - N.V_ab) /dy_n)
/(dy_s+dy_n);

// force source term
F = -rho*g + (B0.P - T0.P)/dz;

// heat exchange between the volumes (conductive heat transfer)
B0.q = A*(B0.T - T0.T)*gamma/dz;

// no heat and mass storage within the element
B0.v = T0.v;
B0.q + T0.q = 0;

// zero equation turbulence model, eddy viscosity (ref: chen xu 98)
MU = (mu + 0.03874*Functions.sqrtReg(Vz^2,1e-8))*rho*dz;
...
other equations omitted for brevity
...
end LayerZ;
    
```

7 Validation

Several tests were performed to validate the proposed models, basically by comparing their outcome with that of CFD models. The verification is made by checking that the sub-zones (that are “large” volumes from the CFD standpoint) yield reasonably accurate averages of the quantities that CFD models evaluate on much finer a spatial discretisation. Other verifications were made against literature models realised with various approaches.

As an example of said tests, a natural convection case is reported, for which the experimental and simulation results of [11] and [16] are taken as reference. The experimental setup in the quoted works is the MINIBAT test cell at CETHIL, described in [1], that consists of a 3,1 m×3,1 m×2,5 m room.

The case here shown has two lateral walls with impressed temperature, one “cold” and one “hot”. The temperature distribution provided by the presented models satisfactorily reproduces experimental data, as can be seen by comparing the steady-state situation shown in figure 5, obtained by linear interpolation and subsequent colour coding, with figure 6 in [16], and is also in good accordance with simulation data pro-

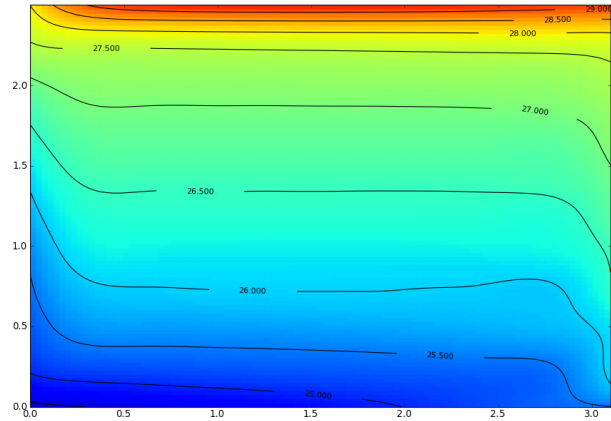


Figure 5: Simulated temperature distribution within a room (°C) with natural convection.

vided by other tools, see e.g. figures 4 and 7 in [16], and figure 5 in [11]. Here too, efficiency is good: on a standard PC, a 3000 s simulation takes approximately 1.5 s only with a 12×10 grid (the same resolution of the quoted references).

8 Application example

To show how the proposed modelling approach allows to efficiently integrate quasi three-dimensional models with one-dimensional ones like for example the piping of a heating system, another brief example is reported. A room (4×1×6 volumes) is heated by a radiator fed with hot water through a three-ways modulating valve, and a PI controller regulates the room temperature. The external conditions are kept constant, and a disturbance is introduced in the form of a sudden drop of the heating water temperature eight hours after the beginning of the simulation. Figures 6 through 8 show the results.

The heater is positioned in the lower left corner, and two tests are reported in which the sensor is located in two different positions. In the first case (“single-Zone1” in the figures) on the same wall as the heater, and above it; in the second one (“singleZone2”) on the opposite wall with respect to the heater. As can be seen, despite the measured temperature is kept at the set point in more or less the same way, the different sensor positions result in different behaviours of the room *mean* temperature, shown in figure 6, thus in different comfort conditions, and energy consumption. Apparently, an analysis like that just sketched would not be possible without the proposed sub-zonal models. Incidentally, the simulation of 24 hours took 1.02 seconds, which is quite good a result.

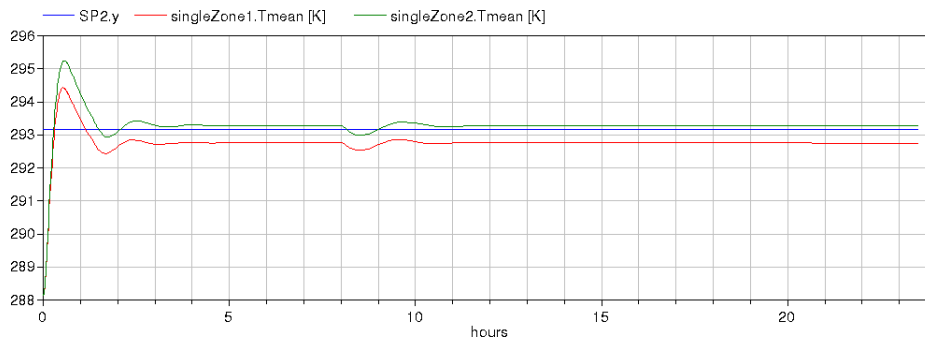


Figure 6: Application example - set point and mean room temperature (not the temperature sensor output) for two sensor locations.

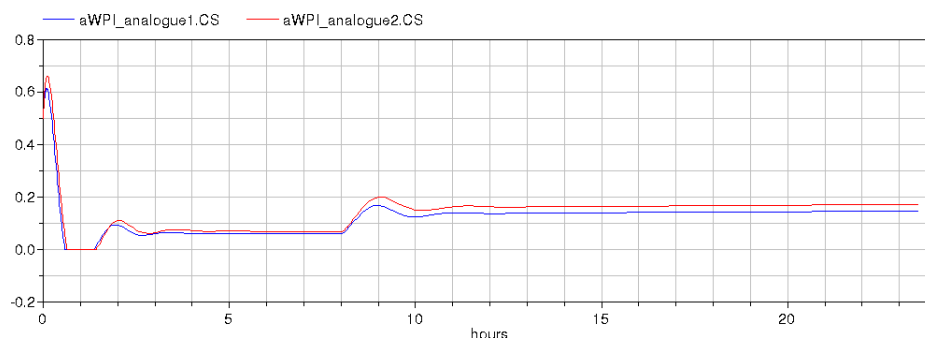


Figure 7: Application example - control signal (heater valve position in the range 0–1) for two sensor locations.

9 Conclusions and future work

An object-oriented modelling approach was proposed to somehow emulate CFD-based results in the context of building simulation. By means of an *ad hoc* equations' formulation and model structuring, high modularity and simulation efficiency can be achieved. Of course the presented models do not fully replicate CFD results, but allow to preserve the relevant facts for energy-related simulation studies. In addition, said models can be readily integrated in a multi-physics environment, thereby avoiding the use of co-simulation to the advantage of speed and model maintenance.

The proposed approach has already demonstrated its validity in terms of modularity, simulation efficiency, and ease of integration with heterogeneous models. This makes the approach particularly suited for system level studies, including (but not limited to) those relative to control.

Future work will be devoted to the representation of complex geometries, and the inclusion of more articulated fluid modelling (e.g., integrating accurate representations of moist air). Further validations will also be carried out, and the obtained models will be progressively integrated, also with others coming from different research lines, in order to construct a general-purpose building simulation library, always keeping in

mind the orientation to system studies, that in the opinion of the authors is one of the major strengths of their proposal.

References

- [1] F. Allard, J. Brau, C. Inard, and J.M. Palier. Thermal experiments of full-scale dwellings cells in artificial conditions. *Energy and Buildings*, 10:49–58, 1987.
- [2] Q. Chen and W. Xu. A zero equation turbulence model for indoor airflow simulation. *Energy and Buildings*, 28(2):137–144, 1998.
- [3] B. Drury and B. Crawley. EnergyPlus: energy simulation program. *ASHRAE Online Journal*, 42(4):49–56, 2000.
- [4] F. Allard F., V.B. Dorer, and H.E. Feustel. *Fundamentals of the multizone air flow model COMIS*. AIVC (technical note 29), 1990.
- [5] F. Felgner, S. Agustina, R. Caldera Bohigas, R. Merz, and L. Litz. Simulation of thermal building behaviour in Modelica. Oberpfaffenhofen, Germany, 2002.

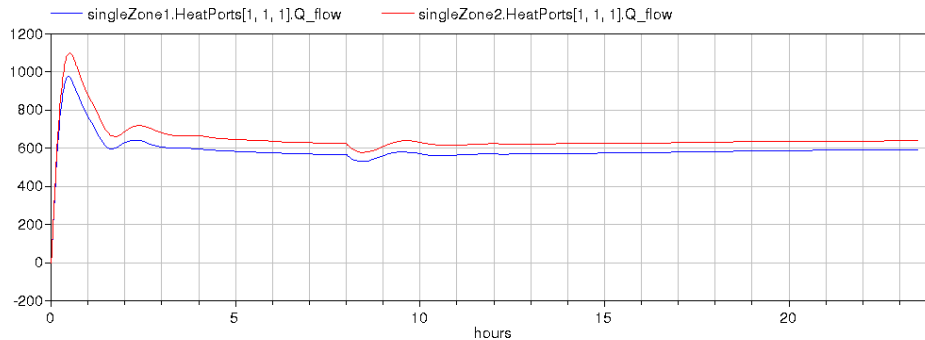


Figure 8: Application example - power released to the room air for two sensor locations.

- [6] F. Felgner, R. Merz, and L. Litz. Modular modelling of thermal building behaviour using Modelica. *Mathematical and computer modelling of dynamical systems*, 12(1):35–49, 2006.
- [7] H.E. Feustel. COMIS—An international multi-zone air-flow and contaminant transport model. *Energy and Buildings*, 30(1):3–18, 1999.
- [8] Fluent Inc. *Fluent 6.1 tutorial guide*. 2003.
- [9] Fluent Inc. *Fluent 6.3 user’s guide*. 2003.
- [10] F. Haghghat, Y. Li, and A.C. Megri. Development and validation of a zonal model - POMA. *Building and Environment*, 36(9):1039–1047, 2001.
- [11] C. Inard, H. Bouia, and P. Dalicieux. Prediction of air temperature distribution in buildings with a zonal model. *Energy and Buildings*, 24(2):125–132, 1996.
- [12] M. Janak. *Coupling building energy and lighting simulation*. Kyoto, Japan, 2000.
- [13] S.E. Mattsson, H. Elmqvist, and M. Otter. Physical system modeling with Modelica. *Control Engineering Practice*, 6:501–510, 1998.
- [14] L. Mora, A.J. Gadgil, and E. Wurtz. Comparing zonal and CFD model predictions of isothermal indoor airflows to experimental data. *Indoor Air*, 23(2):77–85, 2003.
- [15] S.V. Patankar. *Numerical heat transfer and fluid flow*. Taylor and Francis, London, UK, 1980.
- [16] Z. Ren and J. Stewart. Simulating air flow and temperature distribution inside buildings using a modified version of COMIS with sub-zonal divisions. *Energy and Buildings*, 35(3):257–271, 2003.
- [17] A. Sodja and B. Zupančič. Modelling thermal processes in buildings using an object-oriented approach and Modelica. *Simulation Modeling Practice and Theory*, 17(6):1143 – 1159, 2009.
- [18] The Modelica Association. Modelica home page. <http://www.modelica.org/>, 1997–2010.
- [19] H.K. Versteeg and W. Malalasekera. *An introduction to computational fluid dynamics: the finite volume method*. Pearson Prentice Hall, Upper Saddle River, NJ, USA, 2007.
- [20] G.N. Walton. *CONTAM’96 users manual*. NISTIR 6055, National Institute of Standards and Technology, 1997.
- [21] M. Wetter. Multizone airflow model in Modelica. pages 431–440, Vienna, Austria, 2006.
- [22] M. Wetter. Modelica-based modeling and simulation to support research and development in building energy and control systems. *Journal of Building Performance Simulation*, 2(1):143–161, 2009.
- [23] M. Wetter. Modelica library for building heating, ventilation and air-conditioning systems. Como, Italy, 2009.
- [24] D.C. Wilcox. *Turbulence modeling for CFD, third edition*. La Canada, DCW Industries, 2006.
- [25] E. Wurtz, J.M. Nataf, and F. Winkelmann. Two- and three-dimensional natural and mixed convection simulation using modular zonal models in buildings. *International Journal of Heat and Mass Transfer*, 42(5):923–940, 1999.

Numerical coupling of Modelica and CFD for building energy supply systems

Manuel Ljubijankic¹ Christoph Nytsch-Geusen¹ Jörg Rädler¹
Martin Löffler²

¹Universität der Künste Berlin, Lehrstuhl für Versorgungsplanung und Versorgungstechnik
Hardenbergstraße 33, 10623 Berlin

²TLK-Thermo GmbH, Hans-Sommer-Str. 5, 38106 Braunschweig
nytsch@udk-berlin.de m.loeffler@tlk-thermo.de

Abstract

This paper presents an integrated method for the simulation of mixed 1D / 3D system models in the domain of building energy supply systems. The feasibility of this approach is demonstrated by the use case of a solar thermal system: the sub-model of a hot water storage is modeled as a detailed three-dimensional CFD model, but the rest of the system model (solar collector, hydraulics, heat exchanger, controller etc.) is modeled as a simplified component-based DAE model. For this purpose, the hot water storage model is simulated with ANSYS CFD. This detailed sub-model is embedded in the solar thermal system model, which consists of component models of the Modelica library *FluidFlow* and is simulated with Dymola. The numerical coupling and integration of both sub-models is realized by the use of the co-simulation environment TISC. With a comparison of a pure Modelica system model and a mixed 1D / 3D system model of the same solar thermal system, advantages and disadvantages of both simulation approaches are worked out.

Keywords: Co-simulation; Mixed 1D/3D modeling; energy building and plant simulation

1 Introduction

Up to now, the simplified world of DAE (Differential Algebraic Equation) system simulation and the detailed world of CFD (Computational Fluid Dynamics) simulation have been two different “modeling cultures” in the domain of building energy supply systems. Users of component based DAE system simulation tools/approaches like Modelica

[1], MATLAB/Simulink [2], TRNSYS [3] analyze the transient behavior of complex energy system models, which include simplified physical sub-models of the energy supply systems, models for the supplied buildings and models for the control algorithms of the energy management. Because the complexity of these models is reduced (typically some hundred up to 100,000 model variables), the time periods in simulation experiments can be a week, a month or a year.

In comparison to the simplified systems models, detailed 3D CFD models are used to optimized the thermal comfort of a room (e.g. to find suitable positions of inlet and outlet air passages, which guarantee comfortable local air temperatures and air velocities) or to optimize the flow conditions and the heat transfer within a building services component (e.g. to design the inner geometry of a heat water storage). The second type of models uses highly discretized Finite Element Models (FEM) or Finite Volume Models (FVM) with up to several million equations. For this detailed models, the used time period in transient simulation experiments can be - restricted by the currently existing numerical power - some seconds up to some days.

The basic idea of this article is to combine both worlds to a numerical integrated simulation approach for building energy supply systems (compare with Figure 1): the DAE simulation tool (e.g. Dymola) produces with the help of its - component-based - one-dimensional Modelica model transient (“intelligent”) boundary conditions for the detailed three-dimensional CFD tool/model and vice versa. In this way, the most interesting part(s) of a system model can be analyzed on a more detailed level, wherein the system relationships are fully taken into account.

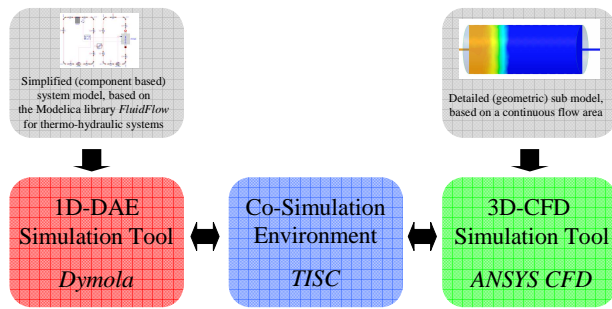


Figure 1: Numerical integrated simulation approach for DAE / CFD modeling of building energy supply systems

The numerical data exchange and the synchronization of the numerical solvers of several simulation tools are realized by the use of the co-simulation environment TISC [12]. In this procedure, it is very important to use “appropriate models” from the DAE- and the CFD-world, this means the models use a comparable physics, and they produce similar results on an aggregated level. Preliminary studies on this have been undertaken by the authors [14, 15].

2 Use case: Solar thermal system

For doing the comparative system simulation studies about the numerical coupling between the DAE-approach of Modelica and the CFD-approach, a solar thermal system for warm water production was used as a reference system (see Figure 2).

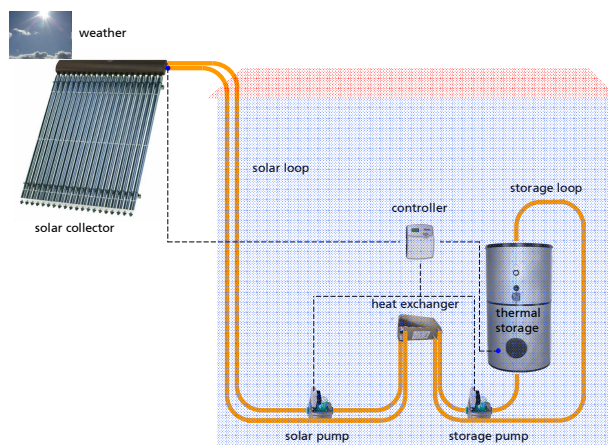


Figure 2: Used solar thermal system for the simulation studies

The most important components of the solar thermal system are an evacuated tube collector (type Viessmann VITOSOL 200 T) with an aperture area of 3.17 m² and a hot water storage with a volume of 400 liter. The roof collector is aligned to the south

and tilted with an angle of 30°. Here, the vertical distance between the roof and the storage is 10 m. The cylindrical shaped storage has a height of 1.45 m and a diameter of 0.59 m and is isolated with 100 mm insulation ($\lambda = 0.06 \text{ W}/(\text{m}\cdot\text{K})$). An external plate heat exchanger ($k\cdot A = 1,000 \text{ W}/\text{K}$) 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 (mass flow rate 0.0264 kg/s), if the collector outlet temperature is 4°K higher than the temperature in the lower part of the storage (hysteresis of 5°K). All hydraulic components of the solar thermal system are connected with copper pipes with an inside diameter of 26 mm, a wall thickness of 1 mm and an insulation thickness of 30 mm ($\lambda = 0.035 \text{ W}/(\text{m}\cdot\text{K})$). For the climate boundary condition Meteornorm [16] weather data from Hamburg (Germany) were used. In the simulation scenario the load process for the thermal water storage over a time period of 24 h (86,400 seconds) during a summer day were analyzed. At the beginning of the load process all the fluid temperatures in the collector, in all pipes and in the storage shall be 20 °C.

3 System simulation with Modelica

To obtain a reference system for the evaluation of the coupled Modelica/CFD-model, in a first step the solar thermal system was modeled as a pure Modelica model. For this purpose, the *FluidFlow*-library was used.

3.1 Modelica FluidFlow library

The Modelica-library *FluidFlow* is being developed at Udk Berlin for thermo-hydraulic network simulation [4]. The main application field of this library is the modeling of solar thermal systems, HVAC (Heating, Ventilation and Air-Conditioning)-systems and district heating/cooling systems.

The *FluidFlow*-library comprises a set of “ready-to-use” standard hydraulic models, such as pipes, elbows, distributors and pumps. Further, the library includes more specialized models from several domains (compare with Figure 3), such as solar thermal technologies (collector models), thermal storage technologies (storage models) or energy transformation technologies (e.g. models of heat exchangers, absorption chillers and cogeneration plants).

The weather data sets are read and interpolated with a new developed Modelica component, based on the ncDataReader2 library, which provides access to external data sets as continuous functions [5].

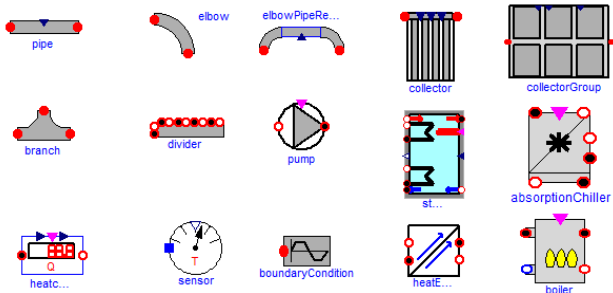


Figure 3: Standard models (left) and specialized models (right) of the thermal-hydraulic Modelica library *FluidFlow*.

Up to now the main application field of the *FluidFlow*-library was the modeling and simulation of complex energy supply systems (heating and cooling energy) for new planned city districts with residential buildings in Iran [6].

3.2 Modeling as pure Modelica system model

Figure 4 shows the system model of the solar thermal system from the use case, solely modeled with sub-components of the Modelica *FluidFlow*-library.

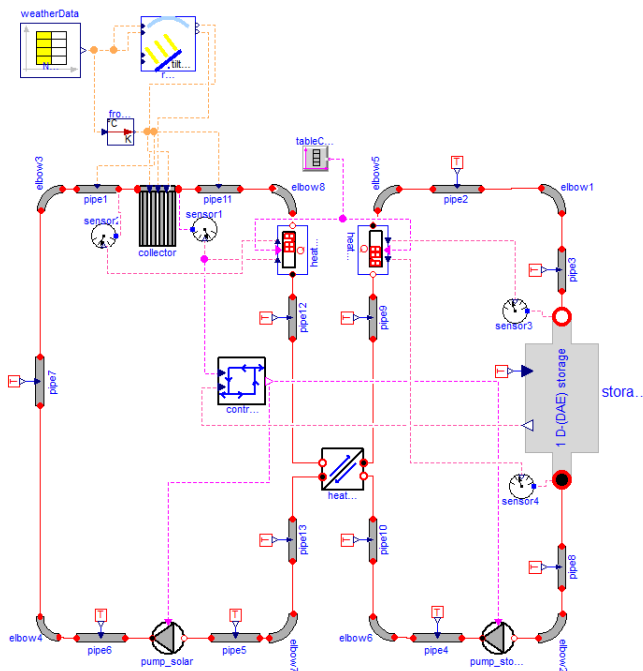


Figure 4: Solar thermal system, modeled as system model with pure Modelica sub-components

The used physical models (solar collector, pipes, external heat exchanger, hot water storage) were validated from [7]. All the pipes were parameterized in a way that a minimum of 1 numerical node per 1 m pipe length can be ensured. The hot water storage

model is divided into 10 thermal horizontal zones. This modeling approach leads to a system model with 721 time variables. With the symbolic reduction algorithm of the used simulation tool Dymola [8], the DAE-system could be reduced to 419 time varying variables.

In order to define suitable and comparable interfaces to the Modelica/CFD system model, a sub-system for the hot water storage and its boundary conditions (connection pipes, inlet boundary conditions, models for the pressure losses for the flow dilation and contraction) were introduced (compare with Figure 5).

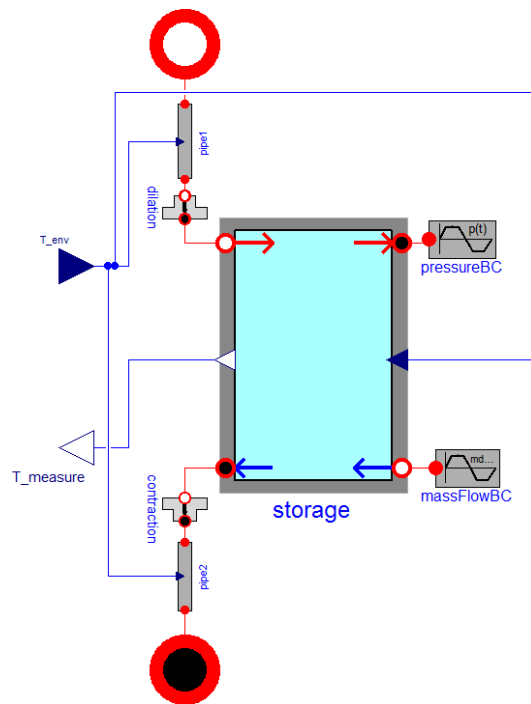


Figure 5: Modelica sub-system for the thermal storage and its boundary conditions

4 Modelica system simulation with an integrated CFD sub-model

4.1 Modeling and simulation with ANSYS CFD

In order to determine the model state at any point of the volume of the hot water storage (e.g. temperatures, velocities, pressures), the three-dimensional CFD method was used. In our case, the fluid region of the hot water storage is modeled with ANSYS CFD Release 12.1 [9], which works with CFD algorithms, based on the **Finite Volume Method (FVM)**. The FVM method calculates approximated solutions of the partial differential equation system, which describes the transport process of momentum, mass and

heat transfer within the flow region (Navier-Stokes equations):

Continuity equation:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{U}) = 0, \quad (1)$$

Three momentum equations (vector notation with the Nabla-operator for the three Cartesian coordinates):

$$\frac{\partial (\rho \mathbf{U})}{\partial t} + \nabla \cdot (\rho \mathbf{U} \otimes \mathbf{U}) = -\nabla p + \nabla \cdot \boldsymbol{\tau}, \quad (2)$$

Total energy equation:

$$\begin{aligned} \frac{\partial (\rho h_{tot})}{\partial t} - \frac{\partial p}{\partial t} + \nabla \cdot (\rho \mathbf{U} h_{tot}) \\ = \nabla \cdot (\lambda \nabla T) + \nabla \cdot (\mathbf{U} \cdot \boldsymbol{\tau}). \end{aligned} \quad (3)$$

For each element of the flow region a discretized form of the mentioned 5 partial differential equations is calculated from the ANSYS CFD numerical solver.

4.2 3D CFD model of the hot water storage

An adequate three-dimensional model of the hot water storage has to fulfill several aspects:

1. The discretization of the fluid regime has to be fine enough to limit the numerical error. For this purpose, the size of the elements has to be adapted to the local geometries und the local velocities.
2. The CFD model has to be fast enough for a coupled transient simulation.
3. The location of the interface planes between the three-dimensional CFD fluid regime and the one-dimensional Modelica fluid regime have to be chosen in a way, that the fluid flow at the interface point can take place without a significant disturbance, which can be induced by the inner fluid flow pattern of the storage.

For this reason, the fluid regime of the cylindrical storage was supplemented by two connection pipes with a length of 0.275 m. This necessary length was established by preliminary CFD flow pattern tests with a typical mass flow from the storage pump (0.026 kg/s). The compromise between a needed accuracy and a desired numerical performance is a

mesh with all in all 21,904 numerical nodes (85,230 finite volume elements). Hereof 5,440 numerical nodes (5,370 finite volume elements) are used for the connection pipe models and 16,464 numerical nodes (79,860 finite volume elements) for the storage model (compare with Figure 6). The used turbulence model was a laminar model, because the range of values of the Reynolds number of the fluid regime, which can be assigned to the connection pipes, reaches from of 1,290 (20 °C) up to 2,230 (47 °C).

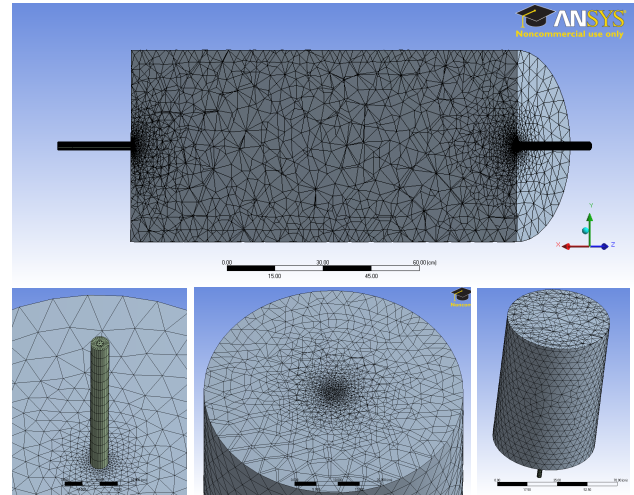


Figure 6: Adapted Mesh of the finite volume storage model: longitudinal section (above), horizontal view on the storage with connection pipe (below left), horizontal sections close to the inlet (below middle) and in the middle of the storage (below right)

4.3 Numerical tool coupling by TISC

TISC [12] - the TLK Inter Software Connector - is a co-simulation environment for coupling different simulation programs. The software is platform independent and uses TCP/IP-sockets for communication. TISC provides interfaces to 1-D and 3-D simulators (like MATLAB/Simulink, Dymola, KULI, CFX, Star-CD) and more abstract interfaces written in C, C++, Java and Fortran (see Figure 7).



Figure 7: TISC simulation environment

Tool coupling has several advantages (see [10]). The best tool can be chosen to describe all parts of a system. Present software and expert knowledge can be used for modeling complex systems from diverse

physical and engineering domains (see [11]). In some cases, it also makes sense to break down a model or system into several sub-systems to decouple the time steps and accelerate the simulation. Using TISC makes it also possible to integrate transient boundary conditions from different specialized simulation tools. For example, the input and output data from a detailed FEM or CFD simulation tool are available for a DAE Modelica model in Dymola and vice versa. Distributed computation of a system on several cores also provides the opportunity to parallelize a simulation calculation and get more computation performance.

The TISC environment is divided in two layers, the Simulation-Layer and the Control-Layer. The Control-Layer is responsible for starting and parameterizing all models at simulation start. It is also possible to configure a set of simulations and let them run automatically one after another in batch-mode. This is helpful for using computer resources efficiently.

After a simulation has started, the Simulation-Layer is responsible for handling data-exchange and synchronization. Using the Control-Layer is optional and a distributed simulation can be performed using only the Simulation-Layer.

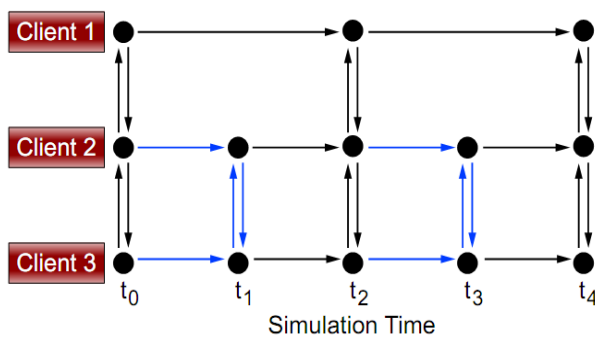


Figure 8: TISC synchronization scheme

TISC supports sequential (“explicit”) synchronization, while parallel (“implicit”) synchronization (see [13] and Figure 8) is often used. When using parallel synchronization all models are being calculated simultaneously. With increasing number of models this leads to a major increase in simulation speed.

TISC uses fixed step-sizes for data-exchange between the models and the server. The synchronization rate of the models has a highly influence on the total simulation time, because every time a model gets synchronized with the TISC server, depending on the used simulation program, an event is generated and a convergent solution for the model has to be found again. With an increasing synchronization rate for system models with relative large time constants,

the number of synchronization events can be reduced and the simulation experiment can be relevant accelerated. It is also possible, that each model uses its own synchronization rate.

4.4 Coupled Modelica / CFD system model

Figure 9 illustrates the coupled Modelica / CFD system model of the solar thermal plant, where the one-dimensional DAE hot water storage model was substituted by a container model, which includes the coupled ANSYS CFD model and the corresponding TISC coupling components.

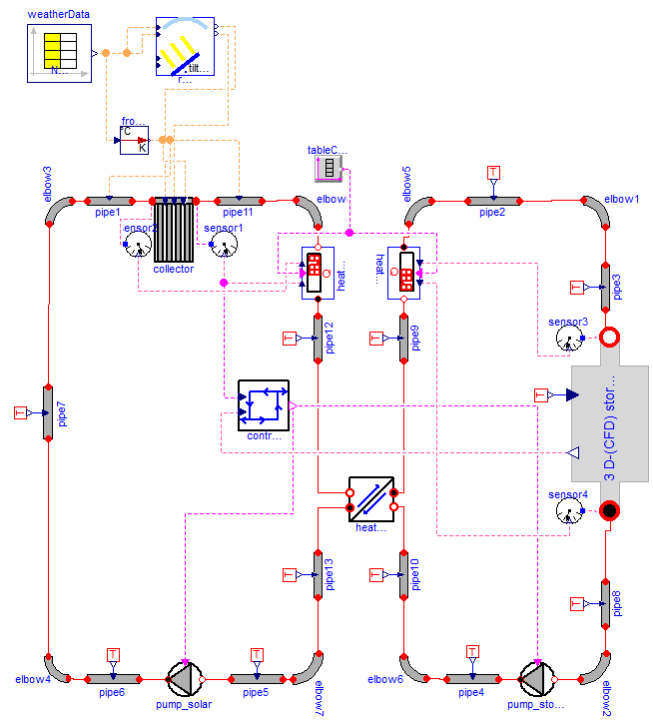


Figure 9: Solar thermal system, modeled as system model with coupled Modelica and CFD sub-Components

Beside the interfaces for the inlet and outlet mass flow, the Modelica / CFD sub-system models has further interfaces to the storage environment temperature and to the temperature sensor within the 3D fluid regime, which is axially positioned 7.3 cm above the bottom of the storage. The hot water storage model has to be exchanged both temperature values with the “Modelica world” to determine the heat loss through the storage insulation and to provide the measured storage fluid temperature as an input for the two point controller model (compare with Figure 10).

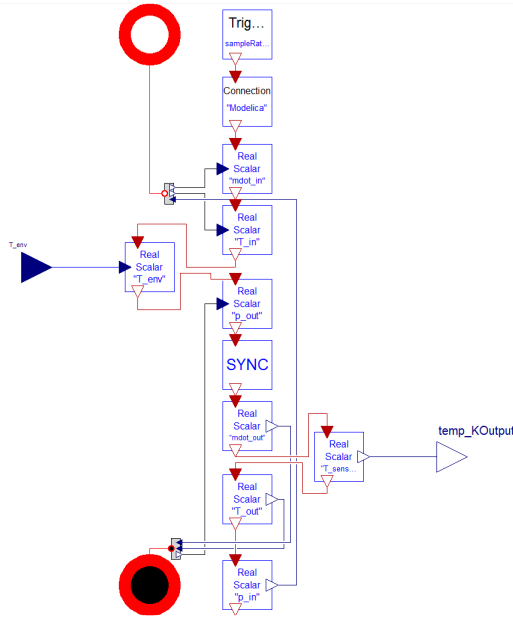


Figure 10: Modelica / CFD sub-system for the thermal storage

5 Comparative results of both simulation approaches

The Simulation experiment for the pure Modelica system model was performed with Dymola 7.4, using the DASSL solver with a tolerance of $1e-4$. The simulation model runs fast and needs only some seconds simulation time for one day real time.

In comparison to this, the coupled Modelica / CFD system model runs approximately half as fast as real time (≈ 50 hours simulation time). For this simulation experiment, an Apple MacPro workstation with 8 Xeon cores (2.8 GHz) and 32 GB RAM with the operation system Linux OpenSuse 11.2 was used. Doing this, ANSYS CFD used 8 cores for the parallelized CFD-simulation and Dymola used a separate core for the monolithic DAE-simulation on another Windows workstation. The synchronization rate of TISC was set on 1 second.

Figures 11 to 14 illustrate the run of the curves for the most important state and process variables of the solar thermal system for a summer day in June (174th day of the year).

Figure 11 shows the beam, diffuse and total solar irradiation on the tilted solar collector. The value for the total radiation exceeds 800 W/m^2 at midday.

Figure 12 illustrates the inlet and outlet fluid temperatures of the solar collector and the outside air temperature for both simulation approaches. The basic running of the temperatures values is similar (temperature levels, switching-on and -off events

etc.), but the values of the system with the CFD-storage model are more differentiated. The most important reason for these differences consists in the non-existent momentum model within the DAE storage model and its restricted space resolution to one dimension.

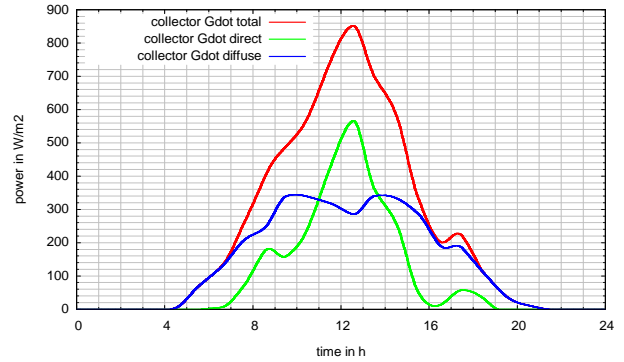


Figure 11: Direct, diffuse, total solar irradiation on the tilted collector

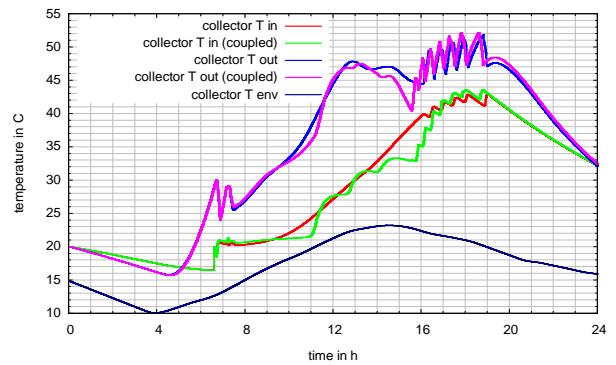


Figure 12: Collector input and output temperature for the pure Modelica system model and for the coupled Modelica / CFD system model

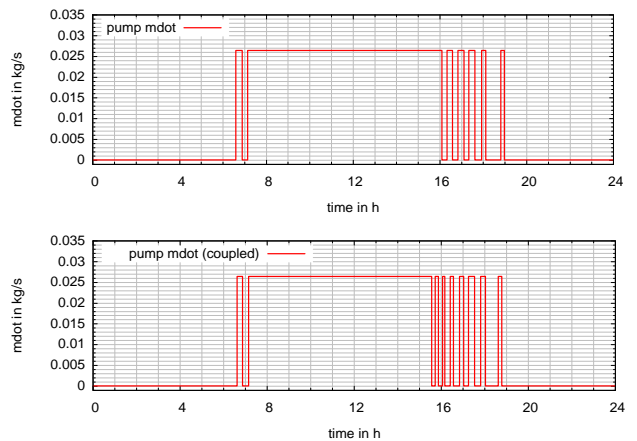


Figure 13: Mass flow of the collector and the storage pump for the pure Modelica system model (above) and the coupled Modelica/CFD system model (below)

Figure 13 demonstrates the switching characteristic of the two-point controller at hand of the timeline of the mass flow, transported by the pumps. The basic pattern of the mass flows for the pure Modelica model and the mixed Modelica / CFD model is similar until 16 o'clock, when the solar irradiation intensity drops significantly. Then, differences in the controller behavior are clearly visible.

Figure 14 shows the inlet and outlet fluid temperatures of the connection pipes between the storage sub-model and the rest of the solar thermal system model (indices *plane in*, *plane out*). Further, 10 representative temperatures within the storage fluid volume are represented. For comparing the 10 vertical temperature values of the pure DAE Modelica model with the detailed temperature field of the CFD model, integrated mean values over 10 horizontal volumes were used (indices *volume 1* to *volume 10*).

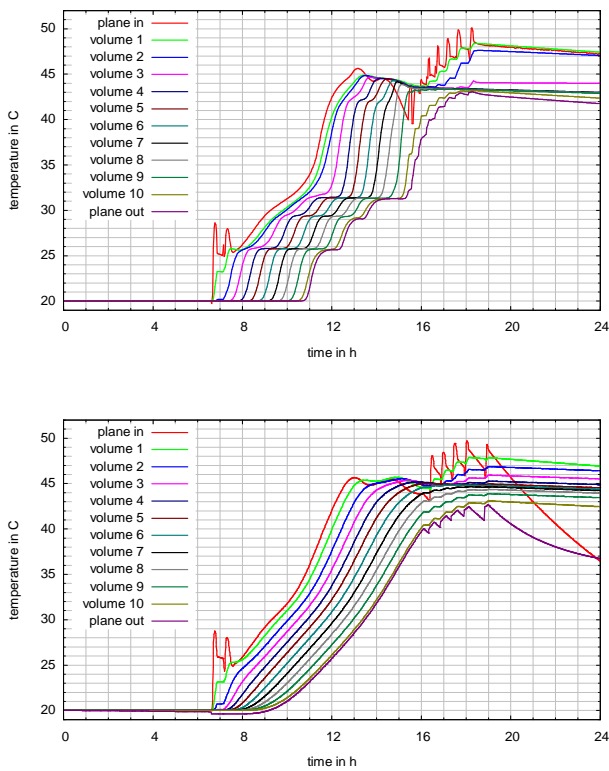


Figure 14: Inlet, outlet and layer temperatures of the thermal storage for the CFD/FVM-model (above) and the Modelica/DAE-model (below)

The temperature levels in both storage models increase during the day parallel to the stored thermal energy. The impact of the switch-on/switch-off characteristic of the mass flows on the storage temperature values developing can be clearly recognized during the morning hours and the evening hours. The CFD storage model shows a significantly more complex behavior: If the stored thermal energy flux

changes or the incoming mass flow switches between zero and its maximum value, the CFD model shows an immediate reaction, because the momentum transport and the natural convection are part of the CFD algorithm.

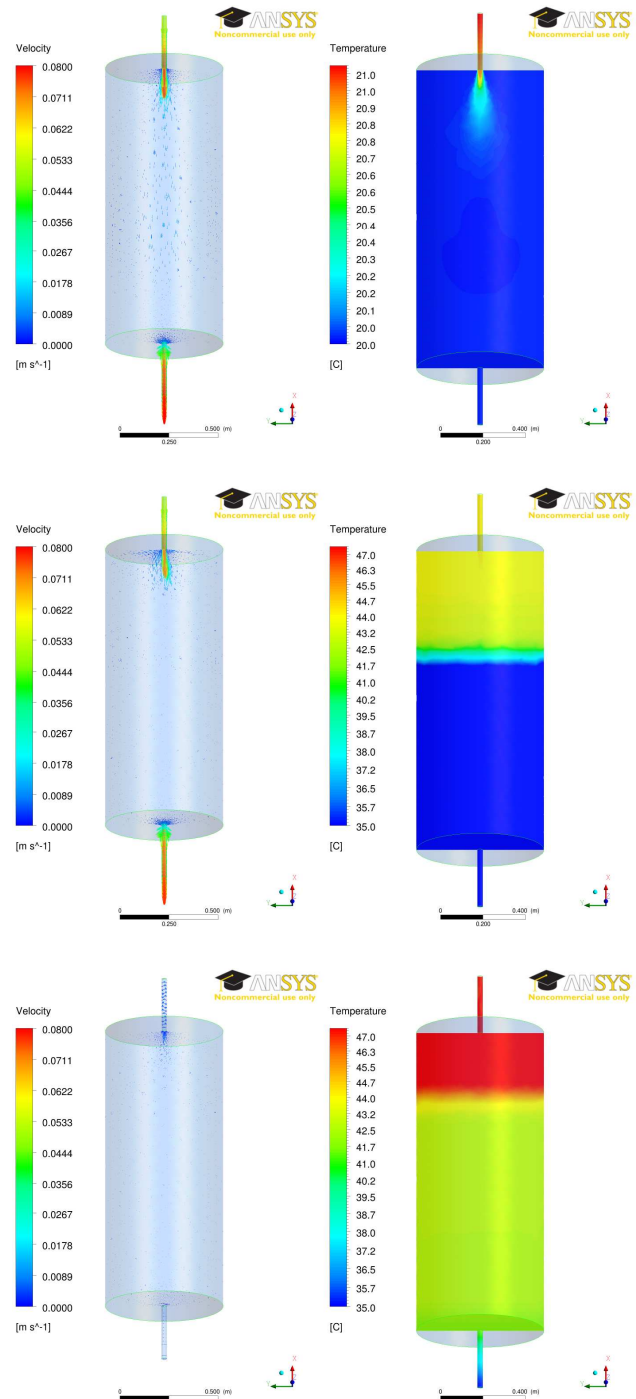


Figure 15: Vertical section of the velocity field (left) and temperature field (right) after the first switch on event at 6:44 (above), at 13:00 (in the middle) and at 24:00, calculated by the coupled CFD/Modelica model

An interesting phenomenon can be observed at the point *plan in* after sunset (\rightarrow mass flow of the storage pump equal to zero): During this time period, the DAE Modelica storage model shows an obvious temperature drop, while the temperature level on the same point of the CFD storage model has only a small decline. The reason for this difference lies in the natural convection effect, induced by the comparatively heat loss effect for the fluid in the small connection pipe in contrast to of the heat loss effect for the fluid within the storage. As a result, the natural convection compensates the increased heat loss of the connection pipe by transporting additional thermal energy from the highest (and hottest) layer of the storage. This effect leads also to the greatest velocity in the region of the inlet connection pipe. The same phenomenon with the reinforced heat loss and the resulting induced convection can be recognized within the vertical sections of the temperature field and the velocity field at the end of the day (24 o'clock, compare with the third picture in Figure 15 and its enlargement in Figure 16).

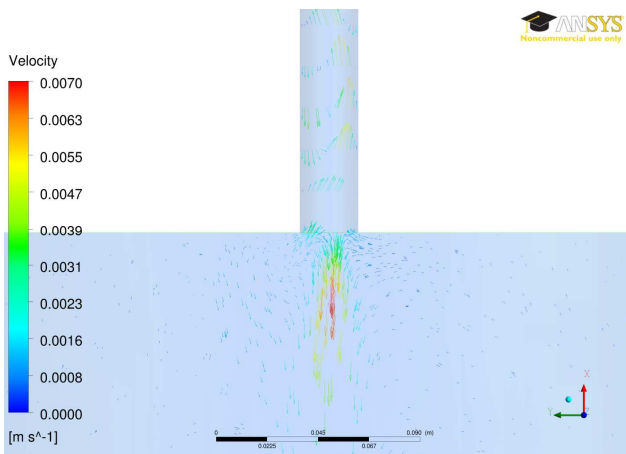


Figure 16: Natural convection phenomenon (velocity field) at the inlet of the hot water storage CFD model at 24:00

Figure 17 illustrates the comparison of the calculated heat energies (supplied heat from the collector model and the inducted heat into the hot water storage) for the pure Modelica system model and for the coupled Modelica / CFD system model as integrated power values. During the whole load process there is only a very small difference between both simulation approaches. The differences between the gained energy from the collector and the inducted energy into the storage are the thermal losses of the hydraulic components.

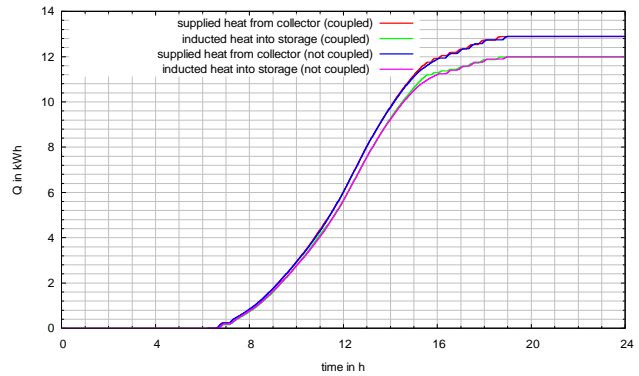


Figure 17: Supplied heat from the collector model and inducted heat into the hot water storage for the pure Modelica system model and for the coupled Modelica / CFD system model

6 Summary and Outlook

It could be demonstrated by the example of a solar thermal plant, that the mixed DAE / CFD simulation approach works. This, the pure Modelica system model showed a qualitatively similar behavior (time-lines of the temperature and of the discrete controller events) and quantitative nearly identical energy values in comparison to the mixed model. In addition, the detailed CFD sub-model of the hot water storage allows analysis for detailed questions (e.g. to find an optimized temperature sensor position or for studying convection phenomena) with full consideration of the surrounding system model. A sufficiently discretized CFD model requires at a up-to-date computer hardware computing times twice as long as real-time.

The next steps of the research will be a detailed analysis of the pressures losses within the different parts of the system (e.g. pressure fluctuations during discrete controller switching events). In addition further system models with more than one CFD sub-model (e.g. a collector CFD sub-model and a storage CFD sub-model) will be considered. For the acceleration of the computation time of the coupled system model, the optimization of the numerical coupling parameters (e.g. the synchronization rate between both simulation tools) and parameter studies with different fine discretized meshes of the hot water storages will be considered.

References

- [1] Homepage Modelica Association: <http://www.modelica.org>
- [2] Homepage MATLAB/Simulink: <http://www.mathworks.com/products/simulink>
- [3] Homepage TRNSYS: <http://www.trnsys.com>
- [4] M. Ljubijankic, C. Nytsch-Geusen: Thermo-hydraulische Simulation solarthermischer Systeme mit Modelica. In Proceedings: 18. Symposium Thermische Solarenergie, OTTI-Technologiekolleg, Regensburg, 2008.
- [5] Homepage ncDataReader: <http://www.j-raedler.de/projects/ncDataReader2>
- [6] Ederer, K.; Huber, J.; Nytsch-Geusen, C.; Seelig, S.; Unger, S.; Wehage, P.: Konzeption und Planung solarunterstützter Energieversorgungssysteme für New Towns im Iran. In Tagungsband: 20. Symposium Thermische Solarenergie in Staffelstein, OTTI-Technologiekolleg, Regensburg, 2010.
- [7] C. Nytsch, M. Poli, T. Schneider: Messtechnische Untersuchungen an einer solarthermischen Versuchsanlage zur Validierung der solartechnischen Modelle der Simulationsumgebung SMILE. In Proceedings: 10. Symposium Thermische Solarenergie in Staffelstein, OTTI-Technologiekolleg, Regensburg, 2000.
- [8] Homepage Dymola: <http://www.3ds.com/products/catia/portfolio/dymola>
- [9] Homepage ANSYS CFD: <http://www.ansys.com/products/fluid-dynamics/cfd/>
- [10] Kossel, R.; Tegethoff, W.; Bodmann, M.; Lemke, N.: Simulation of Complex Systems using Modelica and Tool Coupling. In: Proceedings of the 5th International Modelica Conference, Vienna, Austria, Modelica Association, September 2006, 485 – 490.
- [11] Kossel, R.; Försterling, S.; Tegethoff, W.: Einsatz hybrider Simulationstechnik für die Bewertung mobiler Heiz- und Kühlkonzepte. In: Brill, U. (Hrsg.) ; Haus der Technik (Veranst.): Wärmemanagement des Kraftfahrzeugs VI Haus der Technik, Expert-Verlag, Juni 2008 (Haus der Technik Fachbuch 93). – ISBN 978-3-8169-2820-1, S. 150 – 162.
- [12] Kossel, R.; Correia, C.; Loeffler, M.; Bodmann, M. ; Tegethoff, W.: Verteilte Systemsimulation mit TISC. In: ASIM-Workshop 2009 in Dresden mit integrierter DASS'2009, 2009.
- [13] W. Puntigam et al., Transient Co-Simulation of Comprehensive Vehicle Models by Time Dependent Coupling. In: SAE 2006 Transaction Journal of Passenger Cars: Mechanical Systems, ISBN 978-0-7680-1838-7, pages 1516 - 1525.
- [14] M. Ljubijankic, C. Nytsch-Geusen, S. Unger: Modelling of complex thermal energy supply systems, based on the Modelica-Library FluidFlow. Proceedings 6th International Modelica Conference. In Proceedings: 7th International Modelica Conference, 20./22. September, Como, 2009.
- [15] M. Ljubijankic, C. Nytsch-Geusen: Combining different levels of detail in modelling for an improved precision of HVAC plant simulation. In Proceedings: Building Simulation 2009, International Building Performance Simulation Association, Glasgow, 2009.
- [16] Homepage Meteororm: <http://www.meteororm.com>

A Thermo-elastic Annular Plate Model for the Modeling of Brake Systems

José Luis Reyes Pérez*, Andreas Heckmann* and Ingo Kaiser*

*German Aerospace Center (DLR), Institute of Robotics and Mechatronics
Oberpfaffenhofen, 82234 Wessling, Germany

Abstract

The friction forces generated during braking between brake pads and discs produce high thermal gradients on the rubbing surfaces. These thermal gradients may cause braking problems such as hot spotting and the associated hot judder phenomenon in the frequency range below 100 Hz.

Some consequences of these undesirable vibrations are comfort reductions, a defective braking process, inhomogeneous wear, cutbacks of the brake performance and even damage of brake components.

The present paper proposes a modeling concept that is targeted on this field of application and introduces the new Modelica class `ThermoelasticPlate`, which is implemented in the DLR `FlexibleBodies` library.

Keywords: Disc brake, Modal multifield approach, Thermoelasticity

1 Introduction

Friction braking is necessarily related to high thermal loads which lead to high thermal gradients at the surface of brake discs. It is a known phenomenon that these thermal loads can initiate the onset of unevenly distributed hot spots or bands which in turn results in thermally deformed brake discs [1], [2]. Since the brake pads then slide upon a non-smooth surface while the brake disc rotates, the brake system vibrates, noise is generated and undesirable wear occurs.

Besides experimental studies the finite element method (FEM) [3], [4], [5] [6] or analytical techniques [7] [8] are utilized to analyze the thermo-elastic behavior of brakes in literature. Both methods have advantages and provide valuable results, but both methods are not well suited, if complex scenarios such as the interaction of brakes with suspensions or vehicle control systems are investigated and a system dynamical point of view is adopted.

To this purpose the present paper proposes a novel model of a moderately simplified brake disc. Depending on the user input the thermo-elastic behavior of brake discs is described with approximately 100 up to 1000 degrees of freedom.

The thermal field of the disc is discretized in three dimensions in Eulerian representation, an annular Kirchhoff plate is adapted to evaluate the deformations according to the quasi-static thermo-elastic theory [9, Ch. 2].

In circumferential direction the disc is assumed to be rotational symmetric, in axial direction different layers with different heat capacity and conduction properties and multiple surfaces, where cooling by convection occurs, may be defined.

In order to implement this concept the Modelica class `ThermoelasticPlate` has been introduced into the commercial DLR `FlexibleBodies` library. This paper presents the underlying theory on thermal and thermo-elastic fields, explains the user interface of the `ThermoelasticPlate` class and gives an simulation example. The final section gives an outlook to further efforts in research and modeling of friction brakes and its validation, which is supposed to be initiated by the novel modeling approach.

2 Thermal Field

2.1 Weak Field Equations

In order to describe the thermal behaviour of the brake disc the weak equations for the temperature field $\vartheta(\mathbf{c}, t)$ as functions of the spatial position in cylindrical coordinates $\mathbf{c} = (r, \phi, z)^T$ and time t are deduced from the principle of virtual temperature, see e.g. [10, (7.7)] or [11, (1.3.33)]:

$$\int_V [-(\nabla \delta \vartheta)^T \mathbf{q} + \rho c \dot{\vartheta} \delta \vartheta] dV + \dots \quad (1)$$

$$\dots + \int_B \mathbf{q}_B^T \mathbf{n}_B \delta \vartheta dB = 0,$$

where ρ denotes the density, c the specific heat capacity, dB the boundary element and \mathbf{n}_B the outer unit normal vector. \mathbf{q} symbolizes the heat flux according to Fourier's law of heat conduction depending on the temperature gradient $\nabla \vartheta$ and the thermal conductivity matrix Λ [9, (1.12.16)]:

$$\mathbf{q} = -\Lambda \nabla \vartheta \quad (2)$$

The boundary heat flux q_B may be given explicitly or, if convection occurs, may be specified by the film coefficient h_f and the bulk temperature ϑ_∞ of the fluid [9, Sec. 5.6]:

$$\mathbf{q}_B^T \mathbf{n}_B = -q_B - h_f (\vartheta_B - \vartheta_\infty). \quad (3)$$

2.2 Modal Approach

The discretization of the scalar temperature is performed using the Ritz approximation that allows to separate the thermal field description by a finite-dimensional linear combination of two parts, the first one considers thermal modes and is spatial dependent, i.e. $\Phi_\vartheta = \Phi_\vartheta(c)$ and the second one represent the modal amplitudes and is time dependent, i.e. $z_\vartheta = z_\vartheta(t)$:

$$\vartheta(c, t) = \Phi_\vartheta(c) z_\vartheta(t) \quad (4)$$

The spatial mode functions are formulated using the separation approach of Bernoulli for the spatial coordinates as well, so that (4) may be rewritten as follows:

$$\begin{aligned} \sum_{i=1}^n \Phi_{\vartheta i}(c) z_{\vartheta i}(t) &= \sum_{i=1}^n R_i(r) \Psi_i(\phi) Z_i(z) z_{\vartheta i}(t) = \\ &= \sum_{l=1}^{l_m} \sum_{k=0}^{k_m} \sum_{m=1}^{m_m} R_l(r) \cdot \cos(k\phi) \cdot Z_m(z) \cdot z_{\vartheta i}(t) + \\ &\quad + \sum_{l=1}^{l_m} \sum_{k=1}^{k_m} \sum_{m=1}^{m_m} R_l(r) \cdot \sin(k\phi) \cdot Z_m(z) \cdot z_{\vartheta i}(t), \end{aligned} \quad (5)$$

with $i = 1, 2, \dots, n$, $n = (l_m m_m)(2k_m + 1)$.

According to *Walter Ritz* [12], the trial functions have to be linearly independent and components of a complete system, so that the number of i may be increased as needed in order to improve the approximation. For $R_l(r)$ and $Z_m(z)$ cubic B-splines [13] as

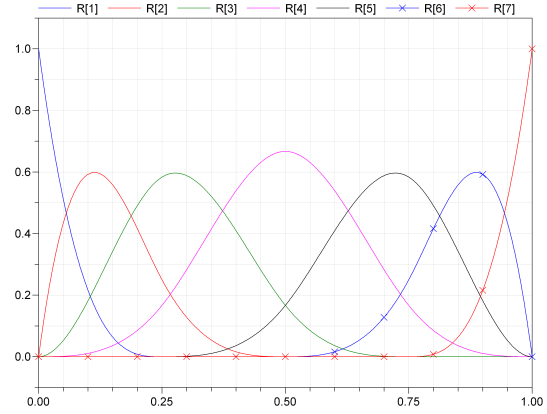


Figure 1: Example set of cubic B-splines to discretize the thermal field in radial and axial direction.

shown in Fig. 1 have been chosen as trial functions in radial and axial direction, respectively.

The harmonic waves (or Fourier series expansion) $\Psi_i(\phi)$ are appropriate in circumferential direction, since

- they allow to represent cyclic properties, i.e. $\Psi_i(\phi) = \Psi_i(\phi + 2\pi)$,
- they are simple to integrate from 0 to 2π ,
- their orthogonality leads to block-diagonal system matrices, i.e. the entire system of equations is split up into decoupled sub-systems,
- they will later on be exploited to provide a Eulerian description of the thermo-elastic plate.

2.3 Discretized Field Equations

If (5) is inserted into (1), the volume integrals can be separated from the terms that dependent on time. As a result, the linear thermal field equation is obtained:

$$\mathbf{C}_{\vartheta\vartheta} \dot{\mathbf{z}}_\vartheta + (\mathbf{K}_{\vartheta\vartheta} + \mathbf{K}_{\vartheta R}) \mathbf{z}_\vartheta = \mathbf{Q}_{\vartheta N} q_B + \mathbf{Q}_{\vartheta R} \vartheta_\infty, \quad (6)$$

where the volume integrals are defined, inter alia using the abbreviation $\mathbf{B}_\vartheta := \nabla \Phi_\vartheta$, as follows [14, Tab. 2.5]:

$$\begin{aligned} \text{the heat capacity matrix:} \quad & \mathbf{C}_{\vartheta\vartheta} := \int_V \rho c \Phi_\vartheta^T \Phi_\vartheta dV \\ \text{the conductivity matrix:} \quad & \mathbf{K}_{\vartheta\vartheta} := \int_V \mathbf{B}_\vartheta^T \Lambda \mathbf{B}_\vartheta dV \\ \text{the Robin load matrix:} \quad & \mathbf{K}_{\vartheta R} := \int_B h_f \Phi_\vartheta^T \Phi_\vartheta dB \\ \text{the Robin load vector:} \quad & \mathbf{Q}_{\vartheta R} := \int_B h_f \Phi_\vartheta^T dB \\ \text{the Neumann load vector:} \quad & \mathbf{Q}_{\vartheta N} := \int_B \Phi_\vartheta^T dB \end{aligned}$$

These volume integrals may therefore be evaluated in advance to the simulation or time integration, respectively.

2.4 The Eulerian Description

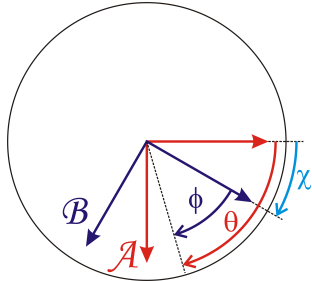


Figure 2: Coordinate transformation with angle χ , that leads from the Lagrangian to the Eulerian description.

It is now considered that the brake disc performs a rotation around its central axis specified by the angle $\chi(t)$. So far the temperature field is described in the so-called Lagrangian point of view [15, Sec. I.3], i.e. the reference frame follows the rotation as it is shown for the coordinate system named \mathcal{B} in Fig. 2.

However for the specific use case treated here it may make sense to resolve the temperature field of the disc in frame \mathcal{A} in Fig. 2. In other words, the observer does not rotate with the disc but looks on the plate from the outside, from a point in rest concerning the rotation with angle $\chi(t)$.

This concept is the so-called Eulerian description [15, Sec. I.4] and is widely used in fluid dynamics, where the motion state of the fluid at a fixed point in space is presented. Due to the rotational symmetry properties of the brake disc the Eulerian description can here be formulated in an elegant and convenient way.

For theoretical derivation the coordinate transformation

$$\phi = \theta - \chi \quad (7)$$

is defined, where θ specifies the angular position of an observed point on the brake disc resolved with respect to the Eulerian reference system \mathcal{A} in Fig. 2.

Furthermore it is assumed that for every trial function in (5) that employs a $\sin(k\phi)$ -term an associated trial function is present where the sinus- is replaced by the cosinus-function only, but $R_l(r)$, $Z_m(z)$ and k are identical, so that mode shape couples c_1 and c_2 exist:

$$\begin{aligned} c_1(r, \phi, z) &= R_l(r) \cdot Z_m(z) \cdot \sin(k\phi), \\ c_2(r, \phi, z) &= R_l(r) \cdot Z_m(z) \cdot \cos(k\phi). \end{aligned} \quad (8)$$

If the following identities

$$\begin{aligned} \sin(k\phi) &= \sin(k\theta) \cos(k\chi) - \cos(k\theta) \sin(k\chi), \\ \cos(k\phi) &= \cos(k\theta) \cos(k\chi) + \sin(k\theta) \sin(k\chi) \end{aligned} \quad (9)$$

are inserted into (8), an associated mode couple $\bar{c}_1(r, \theta, z)$ and $\bar{c}_2(r, \theta, z)$ defined with respect to frame \mathcal{A} appears:

$$\begin{aligned} c_1 &= \underbrace{R_l Z_m \sin(k\theta) \cos(k\chi)}_{:=\bar{c}_1(r, \theta, z)} - \\ &\quad - \underbrace{R_l Z_m \cos(k\theta) \sin(k\chi)}_{:=\bar{c}_2(r, \theta, z)}, \quad (10) \\ c_2 &= \bar{c}_1(r, \theta, z) \sin(k\chi) + \bar{c}_2(r, \theta, z) \cos(k\chi). \end{aligned}$$

As a result of suitable transformations it may also be written:

$$\begin{aligned} \bar{c}_1(r, \theta, z) &= c_2 \sin(k\chi) + c_1 \cos(k\chi), \\ \bar{c}_2(r, \theta, z) &= c_2 \cos(k\chi) - c_1 \sin(k\chi). \end{aligned} \quad (11)$$

The mode functions $\bar{c}_1(r, \theta, z)$ and $\bar{c}_2(r, \theta, z)$ are defined in the Eulerian reference system \mathcal{A} and are linear combinations of the mode functions $c_1(r, \phi, z)$ and $c_2(r, \phi, z)$ described in the Lagrangian frame \mathcal{B} , whereas the combination depends on $\chi(t)$.

This information can be exploited in order to define a transformation: a thermal field resolved in the Lagrangian frame can be transformed to be resolved in the Eulerian frame and vice versa. Of course the physical temperature field itself does not change, but its resolution does so that the numerical values describing the field will be different in frame \mathcal{A} or \mathcal{B} , respectively.

In practice the transformation is formulated in terms of the modal amplitudes $z_{\partial i}(t)$ which are the thermal states in (6):

$$\begin{aligned} \bar{z}_{\partial i1}(t) &= \sin(k\chi(t)) z_{\partial i2}(t) + \cos(k\chi(t)) z_{\partial i1}(t), \\ \bar{z}_{\partial i2}(t) &= \cos(k\chi(t)) z_{\partial i2}(t) - \sin(k\chi(t)) z_{\partial i1}(t). \end{aligned} \quad (12)$$

Again, the new modal amplitudes in the Eulerian frame $\bar{z}_{\partial i}(t)$ are expressed as a linear combination of modal amplitudes in the Lagrangian frame $z_{\partial i}(t)$ and it is just a matter of convenience and practicability in which coordinates the thermal field equations are actually evaluated.

One particularity has been ignored so far. For trial functions with $k = 0$ no mode couple with c_1 and c_2 according to (8) exists, since no associated sinus-function is introduced in (5). As a consequence the transformation (12) is not defined for such modes. However, trial functions with $k = 0$ represent rotational symmetric fields since the dependency on ϕ is

eliminated in (5) due to the term $\cos(k\phi)$. As a consequence mode shapes with $k = 0$ are invariant with respect to rotations with angle χ or in other words: The modal coordinates $z_{\vartheta i}(t)$ related to $k = 0$ are identical in the Eulerian and the Lagrangian description and no transformation is needed.

3 Mechanical Field

The present paper is focused on the thermo-elastic interrelation that rules the behavior of brake discs in frequency range below 100 Hz. Note that there is a complementary paper presented on this Modelica User Conference which is dedicated to higher frequencies in order to cope e.g. with brake squeal phenomena [16]. However here, it is supposed that the excitation is much lower than the lowest natural frequency of the brake disc. In particular the following assumption are made:

- The structural deformations of the brake disc are dominated by its elasticity or thermo-elasticity, respectively, while inertia effects are negligible. The brake disc deforms in a quasi-static manner. This statement is related to the so-called Duhamel's assumption which argues on the different time-scales with which changes in the temperature or deformation field usually proceed, cp. [9, Sec. 2.5].
- A literature review on the characteristics of thermo-elastic brake disc deformation give reason to the assumption that plate bending in some cases even plate buckling is the governing deformation mechanism, see [7], [1], [4]. For example: all experimental studies describe e.g. hot spots to be located alternatively on the two disc surfaces in anti-symmetrical configuration, so that the circumference is deformed similar to a sinuous line. Therefore the deformation field of the brake disc here is represented as an annular Kirchhoff plate.

Note that the description of the annular plate is limited to be linear in this initial implementation, so that plate buckling phenomena are not covered, see [17], [18, Ch. 1]. An extended formulation to consider thermal buckling is a field of active research at the DLR.

3.1 Thermo-elastic Coupling

In [14, Sec. 2.2] the material constitution based on a thermodynamical potential is harnessed to formulate the interrelation of the thermal and the mechanical

field. This approach is not suited here, since the influence of a 3-dimensional thermal on a 2-dimensional displacement field is to describe.

Instead the so-called body-force analogy is employed, i.e. the thermoelastic problem is transferred into an isothermal problem with equivalent distributed body forces σ_{ϑ} [9, §3.3], whose non-zeros components in radial and tangential direction read:

$$\sigma_{\vartheta r} = \sigma_{\vartheta \phi} = -\frac{1+\nu}{1-\nu^2} E \alpha \vartheta, \quad (13)$$

where α denotes the thermal expansion coefficient, E Young's modulus and ν the Poisson number. Together with the relevant strain components in radial and tangential direction ε_r and ε_{ϕ} expressed as functions of the transversal plate deformation w

$$\varepsilon_r = -z w_{,rr}, \quad \varepsilon_{\phi} = -z \left(\frac{w_{,r}}{r} + \frac{w_{,\phi\phi}}{r^2} \right), \quad (14)$$

the associated virtual work δW_{ϑ} reads:

$$\begin{aligned} \delta W_{\vartheta} &= \int_V \delta \varepsilon^T \sigma_{\vartheta} dV = \\ &= E \alpha \frac{1+\nu}{1-\nu^2} \int_V \delta \left(\begin{array}{c} w_{,rr} \\ \frac{w_{,r}}{r} + \frac{w_{,\phi\phi}}{r^2} \end{array} \right)^T \left(\begin{array}{c} z \vartheta \\ z \vartheta \end{array} \right) dV \end{aligned} \quad (15)$$

3.2 Weak Field Equations

The structural displacements u are evaluated on the basis of the principle of virtual displacements [10, (4.7)], which states that the virtual work of the internal forces equals the virtual work of the external forces:

$$\int_V \delta \varepsilon^T \sigma dV + \int_V \delta \varepsilon^T \sigma_{\vartheta} dV = \sum_i \delta u^T f_i, \quad (16)$$

where ε denotes the strain and σ the stress field. f_i represent the applied external forces.

3.3 Modal Approach

Again a Ritz approximation is used to discretize the deformation field u :

$$u(c, t) = \Phi_u(c) z_u(t) \quad (17)$$

The spatial shape functions in (17) are formulated as function of cylindrical coordinates, i.e. $\Phi_u =$

$\Phi_{u,r}, \Phi_{u,\phi}, z$, $w_{,r}$ and $w_{,\phi}$ are partial derivatives with respect to r or ϕ :

$$\sum_{i=1}^n \Phi_{ui} z_{ui} = \begin{bmatrix} -z(\cos(\phi)w_{,r} - \frac{\sin(\phi)}{r}w_{,\phi}) \\ -z(\sin(\phi)w_{,r} + \frac{\cos(\phi)}{r}w_{,\phi}) \\ w \end{bmatrix},$$

$$w = \sum_{l=0}^{l_m} \sum_{k=0}^{k_m} R_l(r) \cdot \cos(k\phi) \cdot z_{ui}(t) + \dots$$

$$\dots + \sum_{l=0}^{l_m} \sum_{k=1}^{k_m} R_l(r) \cdot \sin(k\phi) \cdot z_{ui}(t),$$

with $i = 1, 2, \dots, n$, $n = (l_m + 1)(2k_m + 1)$.

The trial functions in (18) correspond to the trial functions in (5) except of the fact, that a 2-dimensional field is discretized here, while the temperatures depend on all three coordinates.

3.4 Discretized Field Equations

If (18) is inserted into (16) the linear field equation for the displacements is yielded:

$$\mathbf{K}_{uu} z_u + \mathbf{K}_{u\vartheta} z_\vartheta = \sum_i \Phi_{ui}^T f_i. \quad (19)$$

The stiffness matrix \mathbf{K}_{uu} in (19) is defined using the linear displacement-strain operator ∇_u , the abbreviation $\mathbf{B}_u := \nabla_u \Phi_u$ and the elasticity tensor \mathbf{H} :

$$\mathbf{K}_{uu} := \int_V \mathbf{B}_u^T \mathbf{H} \mathbf{B}_u dV \quad (20)$$

The thermo-elastic coupling matrix follows from (15):

$$\mathbf{K}_{u\vartheta} := E\alpha \frac{1+\nu}{1-\nu^2} \dots$$

$$\dots \int_V \left(\Phi_{u,rr} + \frac{\Phi_{u,r}}{r} + \frac{\Phi_{u,\phi\phi}}{r^2} \right) z \Phi_\vartheta dV \quad (21)$$

In addition to the deformations, the motion of the disc's reference frame located at the center of gravity is considered by the Newton-Euler equations [19, (8.6),(8.21)]:

$$m \cdot \mathbf{a} = \sum \mathbf{f}_i,$$

$$\mathbf{I} \dot{\boldsymbol{\omega}} + \boldsymbol{\omega} \times \mathbf{I} \boldsymbol{\omega} = \sum \mathbf{c}_i \times \mathbf{f}_i + \sum \mathbf{p}_i. \quad (22)$$

\mathbf{a} denotes the translational acceleration of the reference frame, $\boldsymbol{\omega}$ its rotational velocity. \mathbf{I} symbolizes the inertia tensor, m its mass. \mathbf{f}_i presents the discrete external forces, \mathbf{p}_i discrete external torques.

4 User Interface

The user interface in Dymola is shown in Fig. 3: here the BrakeForce module represents the input force applied on the brake pads, the PAD module defines the specific locations of the ALE nodes where the one end of the springs will be attached to and provide the kinematics which are fed into the CONTACT module. The CONTACT module is the set of springs and dampers which connects the brake pads with the brake disc.

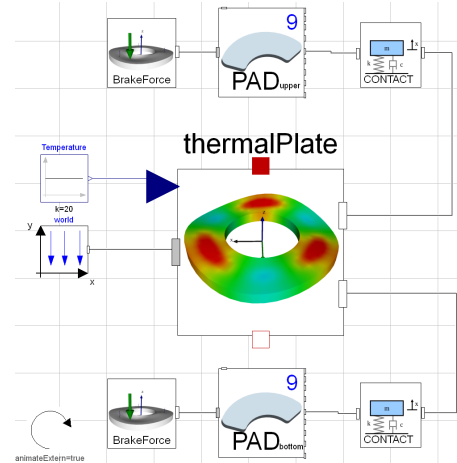


Figure 3: User interface of the thermo-elastic plate.

Finally the thermalPlate module is the block which contains the thermo-mechanical description of the plate which has been derived in the previous sections and also includes the geometrical parameters to model the annular plate (in Table 1).

PARAMETERS	DESCRIPTION
r_i	Inner radius [m]
r_a	Outer radius [m]
th	Thickness [m]
$ksi[:,2]$	Specific points on the plate

Table 1: Geometrical parameters of the plate .

The thermalPlate module contains two types of connectors: the frame of reference and two array frames which represent specific points distributed over the bottom and upper surface of the disc in ALE description. The connectors $nodes_ALE_upper$ and $nodes_ALE_bottom$ are defined by the array ksi . Each row of ksi defines the radial and angular position of one point over the parametrized disc surface contained in the interval $[0,1]$, i.e. if we have $ksi[1,:]=\{0.5, 0.125\}$ the point will be localized in the middle of the distance between the outer radius and the inner radius at 45° in angular position.

In real applications brake discs are subjected to very high increments of temperature during braking which might have a negative impact on the braking performance. In order to reduce the influence of such temperature gradients the so called *cooling channels* are integrated in the structure allowing the air to flow through the mid part of the disc, providing a faster dissipation of the heat transferred to the brake disc. The impact of the cooling channels on the structural dynamics has been also taken into account when modeling the thermo-elastic plate in a simplified way. The idea was to divide the plate thickness in 3 regions (Fig. 4) where the outer regions have a different heat transfer coefficient than the inner region. The calculation of the heat transfer coefficients is not trivial; therefore some assumptions had been done.

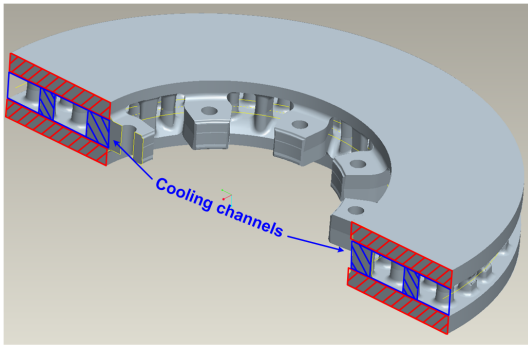


Figure 4: Cooling channels in a brake disc.

Note that conventional frame connectors from the Standard Multibody Library are used within the FlexibleBodies Library, so there is no restriction by connecting other bodies or elements to the *nodes_ALE* connectors. The *thermalPlate* module offers the possibility to select the initial conditions to which the annular plate is exerted and also the discretization parameters that control the modal approach of the annular plate:

- *boundaryConditionRI*: this parameter specifies the boundary condition at the inner radius and provides the options *free*, *supported* and *clamped*.
- *boundaryConditionRA*: this parameter specifies the boundary condition at the outer radius and provides the options *free*, *supported* and *clamped*.
- *radialDiscretization*: this is an integer vector of arbitrary length, in which all nodal diameters numbers to consider have to be given.
- *angularDiscretization*: this is an integer vector of arbitrary length, in which all nodal circles to consider have to be given.

The model has been implemented in the Standard FlexibleBody Library as a complementary example to the already known *Beam* and *ModalBody* classes.

5 Simulation Example

The following example is a simplified representation of a braking system which illustrates an application of the thermo-elastic plate model. The mechanism consists of two brake pads and a thermo-elastic plate. The pads can only perform translations in the direction of the *z*-axis whereas the thermo-elastic plate rotates around the *z*-axis with a constant angular velocity. The geometrical, mechanical and thermal properties of the thermo-elastic plate are listed in Table 2.

PLATE DESCRIPTION	VALUE
Inner radius [m]	0.075
Outer radius [m]	0.15
Thickness [m]	0.022
Density [kg/m ³]	7850
Thermal conductivity [W/m.K]	47
Specific heat [J/kg.K]	70
Thermal expansion coefficient [1/K]	1.04e-5
Young's Modulus [Pa]	2.1e11
Poisson's Ratio	0.29

Table 2: Properties and dimensions of the plate .

During this simulation a force, called normal force, is acting on both brake pads along the axial direction. According to Figure 5 this force is set to zero at the beginning and after certain time step (100 s) the force is increased up to 5 kN. Once this force is applied, the brake pads will tend to move enforcing the contact with the surfaces of the disc and thus the increment of the temperature in the disc due to friction.

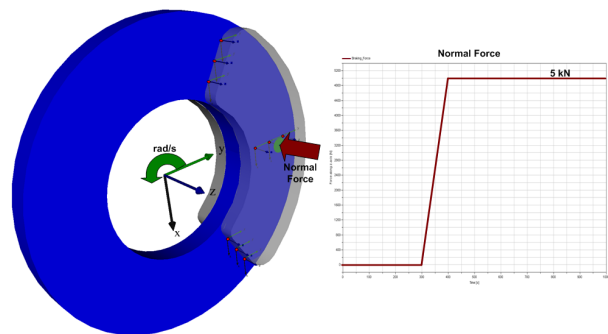


Figure 5: Sketch of the plate with the ALE nodes and normal force.

The contact model consists of a system of dis-

crete linear springs and dampers located at 9 specific nodes on the pads and assembled with its corresponding nodes on the disc. The nodes on the disc are selected according to the ALE formulation (red dots in Fig. 5) and they are fixed in space so that they do not rotate with the disc. At these ALE nodes the forces generated by the springs and dampers are applied to the disc. Due to the sliding friction, heat is generated and it is induced into the brake disc.

Figures 6 and 7 contain the displacement in z-direction and the applied forces of the selected 9 ALE nodes respectively. It is important to observe how the deformation is increased as well as the forces at the ALE nodes as a result of not only the external forces (e.g. normal force) but also internal forces (e.g. thermal stresses) acting in the brake disc.

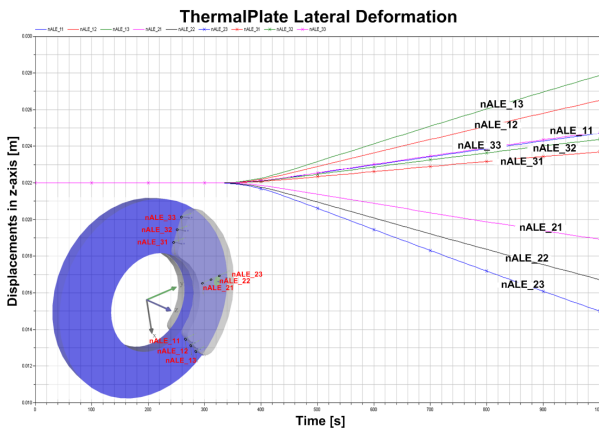


Figure 6: Results of the thermalPlate model for the deformation along z-axis.

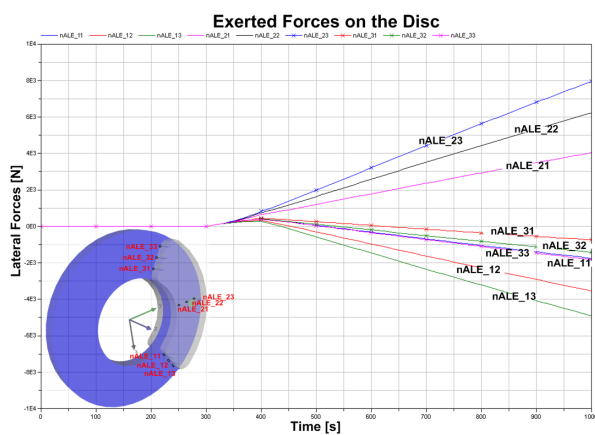


Figure 7: Results of the thermalPlate model for the normal forces acting on the ALE nodes.

As it was explained before the brake disc is rotating with a constant angular velocity. When the pads are in contact with the disc the kinetic energy is trans-

formed into heat which results in an increase of the temperature in the disc. Figure 8 shows clearly the temperature propagation over the whole brake disc. The produced ring-shaped zone has a higher temperature than the rest of the disc due to the contact forces that the disc is exposed to.

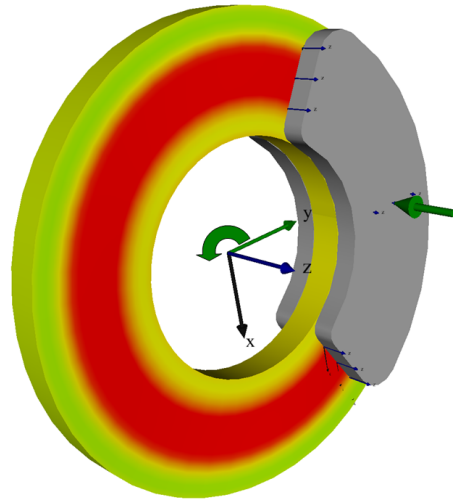


Figure 8: Heat propagation of the thermo-elastic plate with constant angular velocity.

It should be mentioned that these preliminary results still must be validated; nevertheless the results can be interpreted in a physical way and are plausible.

The example presented previously is the representation of a simplified brake disc model, including only braked disc and brake pads, but it represents the basis for a complete modeling of a braking mechanism.

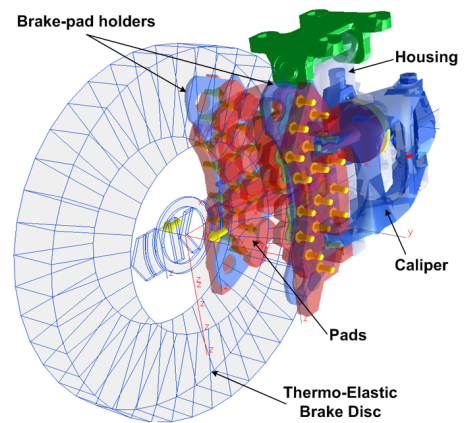


Figure 9: Braking mechanism of a train.

The purpose of this project is to integrate the thermo-elastic model into more complex scenarios, such as: complete braking system of a train (Figure 9) which includes brake disc (thermo-elastic model),

brake pads, rockers, brake pad holders, calipers, housing, brake piston, etc., in order to analyze the induced vibrations, due to the thermo-mechanical deformations, into the complete dynamics of the entire system.

6 Conclusion

The present study investigates the thermo-mechanical effects on the dynamics of a simplified braking system by combining the FEM with the modal approach for flexible bodies in multibody systems. Validation of the presented analysis is still a pendent task; however the results have shown a good agreement with the physical description of this phenomenon giving a solid basis to cope with some of the most common braking problems such as hot spotting.

7 Acknowledgements

The authors highly appreciate the partial financial support of Knorr-Bremse Systems for Railway Vehicles in Munich.

References

- [1] S. Panier, P. Dufrénoy, and D. Weichert. An experimental investigation of hot spots in railway disc brakes. *Wear*, 256:764 – 773, 2004.
- [2] T.K. Kao, J.W. Richmond, and A. Douarre. Thermo-mechanical instability in braking and brake disc thermal judder: an experimental and finite element study. In *Proc. of 2nd International Seminar on Automotive Braking, Recent Developments and Future Trends, IMechE*, pages 231–263, Leeds, UK, 1998.
- [3] A. Rinsdorf. *Theoretische und experimentelle Untersuchungen zur Komfortoptimierung von Scheibenbremsen*. Höppner und Göttert, Siegen, 1996.
- [4] T. Steffen. *Untersuchung der Hotspotbildung bei Pkw-Bremsscheiben*. Number 345 in VDI-Fortschrittsberichte Reihe 12. VDI-Verlag, Düsseldorf, 1998.
- [5] T. Tirovic and G.A. Sarwar. Design synthesis of non-symmetrically loaded high-performance disc brakes, Part 2: finite element modelling. *Proc. of the I Mech E Part F: Journal of Rail and Rapid Transit*, 218:89 – 104, 2004.
- [6] P. Dufrénoy. Two-/three-dimensional hybrid model of the thermomechanical behaviour of disc brakes. *Proc. of the I Mech E Part F: Journal of Rail and Rapid Transit*, 218:17 – 30, 2004.
- [7] K. Lee and J.R. Barber. Frictionally excited thermoelastic instability in automotive disk brakes. *Journal of Tribology*, 115:607 – 614, 1993.
- [8] C. Kremaszky and H. Lippmann. Frictionally excited thermoelastic instabilities of annular plates under thermal pre-stress. *Journal of Tribology*, 127:756–765, 2005.
- [9] B.A. Boley and J.H. Weiner. *Theory of Thermal Stresses*. Dover Publications, Mineola, New York, 1997.
- [10] H.J. Bathe. *Finite Element Procedures*. Prentice Hall, New Jersey, 1996.
- [11] R.W. Lewis, K. Morgan, H.R. Thomas, and K.N. Seetharamua. *The Finite Element Method in Heat Transfer Analysis*. John Wiley and Sons, Chichester, UK, 1996.
- [12] W. Ritz. Über eine neue Methode zur Lösung gewisser Variationsprobleme der mathematischen Physik. *Journal für Reine und Angewandte Mathematik*, 135:1–65, 1908.
- [13] Carl de Boor. *A practical Guide to Splines*. Springer-Verlag, Berlin, 1978.
- [14] A. Heckmann. *The Modal Multifield Approach in Multibody Dynamics*. Number 398 in Fortschritt-Berichte VDI Reihe 20. VDI-Verlag, Düsseldorf, 2005. PhD thesis.
- [15] J. Salençon. *Handbook of Continuum Mechanics*. Springer-Verlag, Berlin, 2001.
- [16] A. Heckmann, S. Hartweg, and I. Kaiser. An Annular Plate Model in Arbitrary Lagrangian-Eulerian-Description for the DLR FlexibleBodies Library. In *8th International Modelica Conference*, 2010. submitted for publication.
- [17] O. Wallrapp and R. Schwertassek. Representation of geometric stiffening in multibody system simulation. *International Journal for Numerical Methods in Engineering*, 32:1833–1850, 1991.

- [18] F. Bloom and D. Coffin. *Handbook of Thin Plate Buckling and Postbuckling*. Chapman & Hall/CRC, Washington, D.C., 2001.
- [19] P.E. Nikravesh. *Computer-aided Analysis of Mechanical Systems*. Prentice Hall, Engelwood Cliffs, New Jersey, 1988.

An Interface to the FTire Tire Model

Volker Beuter
 Kämmerer AG
 Wettergasse 18, D-35037 Marburg
 v.beuter@kaemmerer-group.com

Abstract

The FTire tire model [2] is a well established model for calculating tire forces and torques, especially if the internal dynamics of the tire and higher frequencies have to be considered. This tire model is available for most multi-body simulation programs like ADAMS, SimPack, RecurDyn and for MATLAB. But up to now it was not available in the Modelica world. As the FTire model is only available as binary libraries it could not be ported to a pure Modelica code, but an interface to the existing implementation had to be written. This paper describes the implementation of FTire package making FTire tires available in Modelica (so this paper is rather about interfacing than about tire modeling). The interface involves much more than just a set of Modelica wrappers to C functions: Most callable FTire functions are impure, some having only side effects, no return values. Some considerations have to be taken to ensure the FTire functions are called just often enough. Above this basic Modelica interface layer there is an embedding into the `Modelica.Mechanics.MultiBody` framework with some features beyond FTire itself like non-standard orientation or common definitions for several tires.

The moreover there are some related packages for special interests: The FTireVDL demonstrates the compatibility of the FTire interface to the `VehicleDynamics` package [5] from Modelon. Users of the `Visualization` package [4] from DLR-RM can apply the FTireSimVis package to animate multi-body models with FTire wheels with the SimVis program, exceeding the animation capabilities of Dymola.

Keywords: FTire tire model; interface; MultiBody; visualization

1 Introduction

Part of Kämmerer's involvement in the Eurosylib project [13] was the development of interfaces to ex-

isting tire models. As the FTire libraries, headers etc. were publicly available from the Internet we decided to start with the FTire model. In order to prevent duplicate implementations there was an agreement with Modelon not to implement tire models that had already been implemented (as pure Modelica code) into their `VehicleDynamics` library.

There had been plans to implement an interface to the RMOD-K 7 tire model by Prof. Oertel, FH Brandenburg, Germany as well. Mr Oertel provided useful ideas concerning the anticipated problems caused by the fact that integrating a tire model equipped with its own integrator into a simulation environment always means co-simulation. On his suggestion we implemented a two masses oscillator model, where one oscillator is modeled in pure Modelica the other by an external C function solving its differential equation by means of a simple explicit Euler integrator. But finally Mr Oertel decided to implement a Modelica interface to RMOD-K 7 on its own some day. Meanwhile there was no time any more to start interfacing to another tire model so we concentrated on the interface to FTire.

2 The FTire Tire Model

The FTire tire model was developed by Prof. M. Gipsler, FH Esslingen, Germany and is now distributed by his company COSIN scientific software, Munich [2].

The FTire tire model is available as a stand alone simulator and for most multi-body simulation programs.

The name "FTire" means *Flexible Ring Tire Model*. A tire is conceptually considered as a flexible ring of masses connected by springs in a certain pattern.

But this work is *not* intended to be an introduction into the FTire model. For the matters concerning here, it is only important that a tire mainly interacts with the rest of the model by means of a 3D force and a 3D

torque applied to a certain point in the model and depends on the motion states (position, orientation, velocity and angular velocity) of the wheel center. All the internal structure of the FTire model we can forget here (with the exception of the meaning of some optional additional outputs). We do not build the internal structure of the FTire Flexible Ring model by means of the `Modelica.Mechanics.MultiBody` package, but use the FTire libraries as a set of black boxes to calculate tire forces and some other auxiliary tasks.

It is also important to note that the Modelica interface to FTire does not fit into a decomposition framework of calculating a contact point first, than determining velocities and slip quantities, calculating normal forces and finally other force components [1]. Firstly FTire does not follow this approach; calculations are not based on a theoretical contact point. And even if FTire did so, it is only possible to embed calculations as a whole.

2.1 The FTire Data Files

All the parameters describing tire road interaction in the FTire tire model are in principle stored in two files:

- All tire parameters are stored in a tire property file (default extension `.tir`).
- The road geometry and properties are stored in a road data file (default extension `.rdf`). Some road data file types also refer to further files containing the road geometry.

This means most parameters concerning FTire tires will not directly be provided in the Modelica model but only those two files containing the parameters.

3 The COSIN Tire Interface (CTI) and the FTire Tools

There are several options to access the FTire tire force calculation from a third party software like Dymola in our case. The most favorite way is the *COSIN Tire Interface* (CTI) [3]. It can be used from both FORTRAN and C code, here we use the C versions of the routines. The CTI is available for several operating systems, but we are currently only concerned with the implementation for 32 Bit Windows. The CTI consists of a header file, a small static library (which will be linked to the `dymosim` executable) and a dynamic link library. The static library calls the FTire calculation routines in the `.dll`.

This is all that is needed to access FTire tires from Dymola in Modelica models. But there are some convenient programs from COSIN related to the FTire tire model also used by this interface. The first one, the animation program `COSIN/graphics` consists just of one executable file and directly comes with the CTI.

The moreover there is a whole suite of additional programs, collectively called the FTire tools. Here we especially need the FTire/editor: In principle a `.tir` file is a text file which can be edited with any text editor. But there are two reasons why it is advisable to use FTire/editor:

- FTire/editor provides all appropriate selections. Key words do not have to be memorized and erroneous inputs are reduced.
- In order to speed up simulations FTire does not directly use the input data provided by the user, but does some preprocessing. This preprocessed data is binary and is appended encoded as printable characters at the end of the tire data file. FTire/editor automatically executes a new preprocessing of these data if needed. This excludes simulations based on out-dated preprocessed data.

For FTire version 2010-4 the structure of the downloadable release has been changed. Now the FTire/tools are always included and its version fits to the CTI version. (Earlier versions had release numbers instead of release quarters. The version before 2010-4 was 2.11)

The functions in the CTI generally only provide access to *time dependent* inputs. Most of the *parameters* (in the Modelica sense, i.e. not changing during a simulation) can only be provided in a tire property or road data file. This determined the design of the Modelica interface to FTire: A Modelica model with FTire tires primary refers to `.tir` and `.rdf` files. Most tire parameters have to be changed in the tire property file by using FTire/editor. We do not write any `.tir` and `.rdf` with data entered into a Dymola GUI.

The CTI routines serve several purposes:

- There are data reading functions for the road data and tire property files.
- There are functions which actually do the calculation of the tire forces and torques depending on the current tire positions, velocity, orientation and angular velocity. In principally this is just one function, but there are variants returning the

forces at the (rotating) wheel center or the (non-rotating) hub or with a user defined road geometry. With this minimal set of routines tire simulations can be done already. But in some situations more routines are needed:

- A routine can apply reaction forces to the road part. This is often needed in case the road is not the absolute ground but some moving part, e.g. in test rigs.
- There are routines providing time-dependent inputs in case they are not parameters from the tire property files, e.g. an inflation pressure decreasing during simulation.
- Moreover there are reporting routines returning other calculated quantities than the tire forces and torques. These outputs are either directly returned or are written to special output files. These routines can be used for debugging or visualization purposes, but are also used for the embedding into the MultiBody framework, especially for visualization.

4 The Basic Modelica Interface to the CTI

The lowest level of the Modelica interface to FTire is a set of Modelica functions calling the corresponding CTI routines – directly or indirectly – as external C functions. There is one Modelica function for nearly every function in the CTI. The current version has been written for version 2010-4 of the CTI.

4.1 Calling CTI Routines as External C Functions

Most CTI functions can be directly called from Modelica as external functions. The name of each of the Modelica interface functions is like the corresponding CTI function, but in some cases the argument lists and return values are modified due to Modelica requirements:

- The input argument providing a state is also used as return value. This is done so that this function need not only be used in an empty function call but in an ordinary equation. This is useful, even if the returned variable is just a dummy not used any further: Being used in a real equation apparently ensures that the function is called often enough.

(If used as an empty function call, no calls are performed because from Dymola’s point of view nothing depends on it.)

- Some of the function calls are formally not time dependent (although they are to return time-varying quantities) and Dymola does not evaluate the function over time it seems when the original argument list is used: The returned array is constantly the zero array. By introducing a time dependent variable (time itself) the function is evaluated as expected.
- A similar situation occurs at CTI functions where an input in general is time dependent, but it may also be a constant. Therefore an additional optional input for the simulation time (not passed to the CTI function) is introduced in order ensure a time dependency to prevent Dymola from optimizing away the function call.

Some of the functions here do not have a return value. Their only purpose are their side effects, like turning on or off verbosity. Usually they are only needed once in the beginning of the simulation in order to set a certain mode. A call to them ought to be made as an empty function call in an initial algorithm section.

4.2 C Wrapper functions around CTI routines with function pointer arguments

In some CTI functions added in the last versions there are pointers to functions in their argument list. One example for such a function is a tire force calculation routine for a custom road model. Here a pointer to a road evaluating function is one function argument. (The road evaluation function returns among others the height z at a given location (x,y) on the road at a time t .)

Functional input arguments to functions are not supported in Modelica before version 3.2 and therefore not available at the time of writing this article. In order to support these CTI functions in the Modelica – FTire interface yet, C wrapper functions have been written for them. Such a wrapper has the same argument list like the CTI function in question, only the argument for passing the pointer to the user function (like the road evaluation function) is left out. The wrapper function only calls the CTI functions with the passed arguments and a pointer to an implementation of the user subroutine. The wrapper function is interfaced as an external C function in Modelica in the usual way. By this method also the CTI functions with function

pointers can be supported in the interface but currently the user functions have to be provided in C.

```
// wrapper around CTI function

#include "cti.h"
#include "UserRoadModel.c"

void ComputeForcesWithExtRoad(
  int ti, double t, double* r,
  double* a, double* v, double* w, int mode,
  double* f, double* m, int* ier)
{
  ctiComputeForcesWithExtRoad(
    ti, t, r, a, v, w,
    UserRoadModel, mode, f, m, ier);
}
```

In principle this is all what is needed to include FTire tires into Modelica models. There are some test models where motion states of the tire are calculated and directly provided to the force calculation functions, though.

Parallel processing versions of the CTI functions for calculating tire forces are not yet supported (cf. section 10.3).

Currently no external objects are used in interfacing to FTire, but needed actions like reading tire and road data files are done explicitly. Using a constructor function of an external object may be a cleaner way to do so.

5 The FTire Wheel Model in the MultiBody Framework

But of course this is no convenient modeling of a tire from the user's point of view. A wheel ought to be a component which can be connected to other MultiBody components just by a connect equation to a Frame connector. In principle this means the tire position and orientation from the Frame connector and its time derivatives are passed to the force calculation function. The force and torque variables of the frame connector in turn are equalled to the returned forces and torques. But there are some complicating factors:

5.1 Adjustment of the tire mass

When calculating tire forces FTire only considers the share of the tire mass which is *not* connected rigidly to the rim. The remaining share has to be considered explicitly by the simulation program when calculating inertia forces. This share of mass and inertia is reported by some CTI function. The moreover from

the user's point of view when modeling a vehicle a wheel with a tire ought to be handled together as simple as possible. Therefore the main component of the FTire package is not a *tire* but a *wheel* model also containing a wheel mass. The share of the tire mass and inertia considered fixed to the rim (and therefore not accounted for by FTire) is automatically added to the user provided wheel mass.

5.2 Road Orientation

In FTire itself the orientation of the road is fixed: Global z is pointing up, global x is pointing forward. (This is called the **FTire initial frame**.) This orientation would impose a rather strict restriction on modeling vehicles with the FTire package. The moreover it does not match the default orientation used in the MultiBody package where global y is pointing up. It was a design goal that with the FTire package the road (and therefore the tire) can be oriented arbitrarily so that any existing model can be equipped with FTire tires without the need to re-orient the complete model.

A road orientation can be specified easily by two direction vectors forward and up. (The defaults forward = {1, 0, 0} and up = {0, 0, 1} mean that the usual FTire road orientation is used; up = {0, 1, 0} means the usual MultiBody orientation.) These direction vectors determine a rotation object from the world to the road frame. FTire expects the tire motion states (position, orientation, velocity and angular velocity) resolved in the FTire initial frame. The Frame connector provides the position and orientation resolved in the world frame and the angular velocity in the local frame (the translational velocity can be derived from the position). So first the angular velocity is resolved in the world frame by means of the orientation object of the connecting frame, then all quantities are resolved in the FTire initial frame using the orientation object from the world to the road frame. The force calculation function returns forces and torques resolved in the FTire initial frame. So here they are first resolved in the world frame and then in the local frame, because this is what has to be provided to the connecting Frame. All these calculations and transformations are capsulated into a FTireForce model.

5.3 Tire Orientation

In other multi body programs the orientation of joints and also tires is only determined by the orientation of the connecting frames. But in the MultiBody package the joint axes are defined by some direction vec-

tor(s) (resolved in the local frame). So we do not define the tire spin axis directly by the local frame (like other multi body programs do, taking the local z -axis) but by direction vectors `spin` and `tire_up`. (The vector `tire_up` is only needed for uniquely determining the wheel orientation by providing a reference for the rotation angle.) The defaults `spin = {0, 1, 0}` and `tire_up = {0, 0, 1}` mean that the wheel rotates around its local y -axis. When the connector frame is not rotated relative to the world frame the tire can roll along the global x -axis. In case of the usual `MultiBody` orientation (global y is pointing upwards, i.e. road orientation vector `up = {0, 1, 0}`), `spin = {0, 0, 1}` ought to be used.

Internally the tire orientation is modeled by a `FixedRotation` component. With the default tire direction vectors its orientation is the null rotation.

5.4 Common Tire and Road Properties

In `FTire` itself all tire instances in a model are completely independent, i.e. they all have their own tire property and road data file definitions. But except for some test rigs and special situations all wheels of a vehicle will run on the same road. This road will have the same orientation for all tires and they in turn will have the same orientation. The moreover in many cases there will be the same tire for all instances, i.e. the same tire property file.

In order to prevent the need for providing these property files and the orientation vectors four or more times for a vehicle a common properties model using the `inner / outer` mechanism has been introduced: Each tire model accesses a common property component as outer object. In case a top level model containing an `FTire` wheel model does not contain such a common properties component a default component is used. The values from a common properties object can be overwritten by a wheel model: In case a tire property or road data file string is provided (i.e. it is not empty) or direction vectors are specified (i.e. they are not zero vectors) they are used overwriting the values from the common properties object. In this way it is possible to make exceptions, e.g. providing road data and tire property file by way of the common property object, but using a different tire property file for one wheel in order to model a defect tire.

In case all property files and direction vectors are directly provided at all wheels the common property object is not used at all. When there is only one wheel in the model (like in a tire test rig) this is the easier method.

5.5 Road Part Model

In `FTire` the road does not have to be associated to the absolute ground but tire contact may be calculated towards some movable road part. The `FTire` interface package supports this feature by a model `RoadPart`. Like the wheel models it possesses a usual `MultiBody` Frame connector. The road part model consists of a `Body` component to model its mass and inertia, a road surface visualizer object and a `RoadForce` component.

The `RoadForce` model queries the road part motion quantities (position, velocity, orientation and angular velocity) and provides it to the `FTire` kernel by means of a `CTI` function. When tire forces for the associated wheel are calculated this not done based on the motion relative to ground but on the relative motion to its road part. Therefore each road part needs to have a unique ID corresponding to a wheel. (When the same road part is to be used for more than one wheel a `Body` component (and a road visualizer if needed) can be used together with one `RoadForce` component for each of the wheels running on that part.) The reaction forces from the tire contact are applied to the road part. (Because of the tire inertia this forces do not equal the forces to the wheel with opposite signs.)

Here again it has to be considered that `FTire` itself has a fixed orientation of the road whereas the `Modelica` model using the `FTire` interface package may use some other orientation. Thus the motion states of the road part resolved in the world frame are transformed into the `FTire` initial frame before passing to the `FTire` kernel. Conversely the `CTI` function returns the reaction forces also in the `FTire` initial frame. They are first transformed to the world frame before applying them to the frame connector of the `RoadForce` component resolved in this frame.

6 Tire and Road Visualization

For a tire model based on the theoretical contact point concept a simple cylinder (with some mark to indicate rotation) is all what is needed for tire visualization seen from a technical point of view. The unloaded radius and the tire width (and possibly the radius of the wheel rim to indicate too deep penetration of the tire into the road) is all that is needed. Everything else is a matter of nice animations, but does not bring much new insights into the tire behavior.

Things are different when the internal dynamics of the tire is also considered and the tire (and possibly also road) deformation during simulation is also avail-

able. In this case the animation of the deformed tire is a good plausibility check on the tire simulation.

The CTI provides tire shape information by means of the function `ctiPutNodePositions`. The name may be a bit misleading here. It does *not* deliver the position of one of the internal belt elements constituting the flexible ring modeling the tire. Instead, for any point on the tire surface uniquely characterized by an circumferential angle (scaled to the range $[0, 1]$) and a value for the position along the cross section (also in the range $[0, 1]$) the 3D position of that point is calculated. The position is returned in several coordinate systems. We only use the position in the FTire initial frame.

When the `ctiPutNodePositions` function is evaluated on a regular grid over $[0, 1] \times [0, 1]$ this constitutes a representation of the tire surface. It is important to note that the returned positions are variables, not just parameters. The calculated grid is a representation of the dynamic tire deformations during a simulation. Here it has to be considered that the FTire initial frame is not necessarily the MultiBody world frame, so the surface coordinates have to be resolved in the world frame using the rotation object from the world frame to the FTire initial frame.

Things are similar for the road shape representation: There is a CTI function `ctiEvaluateRoadHeight` calculating the current height of a location (x, y) of the road. Again with a grid of equally spaced evaluation points on a range $[x_{min}, x_{max}] \times [y_{min}, y_{max}]$ the road surface is determined. Usually the surface of a road is static, it is calculated only at initialization time. By a Boolean parameter it can be specified that the road surface is (possibly) time-varying and has to be evaluated at every simulation step. Interestingly the formulations for the surface grid values are the same except for that in the time-varying case the simulation time variable `time` is passed to the road height evaluation function, in the default case the constant 0. Dymola infers that in the second case all the inputs to this function are constants or parameters and therefore it does not need to be evaluated again during continuous integration.

There are two ways these surfaces can be visualized: The model is animated in the Dymola animation window, tire and road are visualized by surface visualizers from Kämmerer's Visualizers package. Alternatively the Visualization package with the external viewer SimVis from DLR-RM can be used like described in the last subsection of this section.

6.1 Kämmerer's Visualizers package

The Modelica Standard Library (MSL) contains visualizer models for animation of geometrical primitives like boxes, cylinders and spheres in Dymola. All these visualizers are based on the model `MB.Visualizers.Advanced.Shape` (respectively `ModelicaServices.Animation.Shape` in Modelica 3.1). Unfortunately this model is not able to animate so called polylines (connected sequences of straight lines) or surfaces. But in the Dymola distribution with the MSL come some tiny Modelica models for visualizers, also for polylines and surfaces. From these ideas a package for visualization of (possibly) time-varying geometric primitives has been developed. In particular it contains several visualizer models for surfaces¹ and a simplified version of the standard visualizer form the MSL applicable in some situations.

The surface geometry is defined by a matrix of 3D positions, i.e. by an input `SI.Position[3] grid[m, n]` where `m` and `n` are Integer parameters. The color and reflectance of the surface can be provided by inputs like at the usual MSL visualizer for geometrical primitives. (This means there is an overall color for the surface. Different colors for a surface can only be realized by splitting up a surface into several sub-surfaces.)

The most simple surface model is `WorldSurface` where the `grid` input defines the surface in world coordinates. The model `Surface` additionally has a position input `r` and an orientation object `R` constituting a frame. Here the surface is defined in this frame. (Compared to the usual MSL `Shape` model here we abstained from additionally defining direction vectors defined in that frame to further orient the surface.) The model `FixedSurface` has a frame connector. It uses a protected `Surface` component as a sub-model where the position and orientation inputs are simply the ones from the enclosing `FixedSurface` model. The sub-model is conditionally disabled dependent on an animation flag. This is the same approach as with the MSL `FixedShape` model. But there is also a version `FixedSurface2` where there is no `Surface` sub-model, but the model is extended from it, simply modifying the position and orientation inputs. In using this method there is no possibility to switch off animation by means of an animation flag. But on the

¹A similar visualizer is now available in the latest version 3.2 of the MSL with Dymola specific implementation in the `ModelicaServices` package to be used in Dymola 7.5. In contrast Kämmerer's `Visualizers` package can also be used even with the MSL 2.2.2 and Dymola 6.1!

other hand if this is not needed, this prevents a duplication of model parameter and variables to two model levels.

The polyline model family is implemented in an analogous way. There is also an alternative Fixed-Shape model extended from the Shape model instead of using it as a subcomponent.

The Visualizers package is used in the FTire package for road and tire visualization.

6.2 Tire and Road Visualization with the Visualizers package

The road visualizer model consists of a FixedSurface2 component where the surface is the grid with the calculated heights. This surface is firstly oriented in the FTire initial frame. In order to account for the correct road orientation the surface visualizer is connected to the outside connector via a FixedRotation component with the orientation calculated from the road orientation vectors and zero translation.

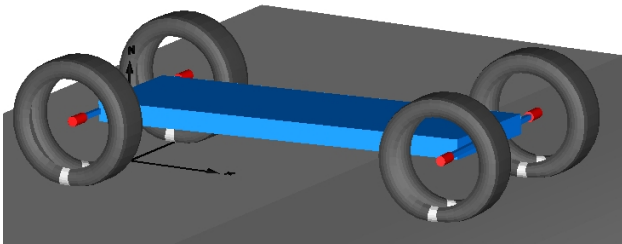


Figure 1: Tire and road visualization using Kämmerer's Visualizers package

The common properties and the RoadPart models already contain a RoadVisualizer object. Therefore a user only has to include a road visualizer into a model explicitly when none of these models is used, i.e. if the road is belonging to the absolute ground and tire property and road data files are specified at the wheel components.

Using this dynamic tire visualization directly within Dymola supersedes to use a separate visualization of the tire with the COSIN/graphics program. Although calculating the dynamic tire shape costs CPU time it is still faster than separate animation.

6.3 Using the Visualization Package instead

DLR-RM has implemented a commercial Modelica package "Visualization" [4] for advanced visualization of multi body models. It does not use the built-in visualization capacities of a Modelica environment

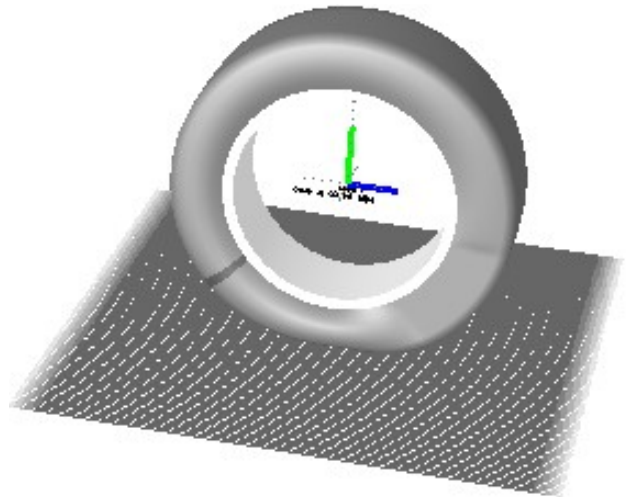


Figure 2: Tire animated with FTire animation tool COSIN/graphics

like Dymola but animates a simulation in a separate program called SimVis. It provides a lot of additional animation features like atmospheric effects, lights and cameras [7]. In particular it also contains visualizer for arbitrary, time-varying surfaces. So this package seemed to be suitable for tire animations.

In order to investigate this a separate package FTireSimVis (using the FTire package) has been implemented. Currently it contains independent tire and road visualizer models based on the Visualization package and an extended wheel model using this tire visualizer and also an extended version of the common properties object with the road visualizer for SimVis. In some later release of the FTire and FTireSimVis packages there ought to be partial visualizer models for tire and road in the base package. The tire and road visualizers in the FTire and FTireSimVis packages will be extended from these partial visualizers. All models in the FTire containing visualizers will declare them as replaceable. The FTireSimVis versions of the models will only have to redeclare the visualizers.

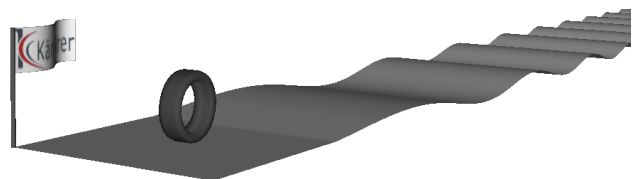


Figure 3: Tire and road animated with SimVis viewer

7 Embedding into the Vehicle Dynamics Library from Modelon

Modelon developed and maintains a commercial package `VehicleDynamics`, also called VDL (for vehicle dynamics library) [5]. It provides a set of tire models completely implemented in Modelica. But it did not include any tire models which involve interfacing to external libraries.² So it is quite a natural question, if the Modelica FTire interface discussed here fits into the VDL framework.

To answer this question a new package `FTireVDL` has been created, because it is not relevant to the basic interface and the FTire package has to stay independent from the VDL. The `FTireVDL` package mirrors the structure of sub-packages as far as needed. There are sub-packages for several layers of vehicle parts, like vehicle, chassis and wheel.

The implementation of the FTire wheel for the VDL package extending from the partial wheel interface model `Conventional` is quite simple: An FTire wheel component from the basic FTire package had to be connected to the outside connector. As the VDL uses another connector (containing besides the usual `MultiBody` frame also a rotational flange to describe the rotation) a `MultiBodyMount` component had to be placed in between. Additionally most of the summary variables could be filled by the TYDEX output variables of the FTire wheel. When some quantity is not available the summary variable is constantly zero.

As the FTire wheel has its own visualization, the tire visualizer from the VDL is not needed. Some of its nice features, like showing contact of the tire to the ground by changing the color of the tire or force vectors (but in this case located in the wheel center, as there is no contact point at FTire) could be added easily in some further version.

Regarding test models there are versions of the "GettingStarted" Sedan car. To this purpose there is a version of its chassis equipped with FTire wheels and also a vehicle model version using this chassis in turn; all the other components are used like in the VDL `SedanTEKBakker` vehicle. (So if there, in the chassis and in the `GettingStarted` experiment model the components had been redeclared replaceable, no new FTire model versions for the chassis and the vehicle had been required and we could simple extend the experiment model and do the required redeclarations.) There is also a simple tire test rig model. Both experi-

ments are available in two versions: In one version the required tire property and road data files are directly provided in the wheel component. An explicit road visualizer component is used to animate the road. The other variant uses an FTire common properties object to specify tire and road. Here we can use the road visualizer contained in that object.

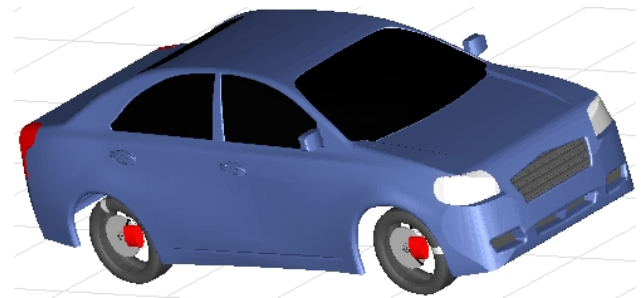


Figure 4: Sedan car from the VDL equipped with FTire wheels

A remark on the used integrator method: Usually FTire preferably works when the solver of the calling simulation tool uses a fixed step size integrator. This also holds for the FTire package in Dymola: We had the best results with the `RKFix4` Integrator. Surprisingly when running the examples in the `FTireVDL` package the simulation fails due to numerical instabilities of FTire itself. (The corresponding VDL models with one of its own tire models works fine with a fixed step size integrator.) This phenomenon deserves further investigation.

In contrast to the FTire package the current `FTireVDL` package is not to be seen as a package ready to use, but as a demonstration that the integration of FTire tires into the VDL framework using Kämmerer's FTire is possible. As the VDL is encrypted and some of its models are not readable on the text layer, a full integration will only be possible in close cooperation with Modelon. The VDL contains a road builder. It ought to be enlarged with the facility to create road data files.

8 The FTire Interface in other Modelica Environments

The FTire package and all related packages have been developed and tested with several Dymola releases. In order to use them with other Modelica environments there are several central demands to that environments. With decreasing importance these are:

²Meanwhile there is an interface to the Delft tire model [6].

- Ability to call C functions by way of the external keyword.

Without this feature no interfacing is possible at all. If at least it is possible to call FORTRAN subroutines the interface can be changed so that the FORTRAN version of the CTI is used instead of the C version.

- (At least partial) support of the `Modelica.Mechanics.MultiBody` package. The FTire interface packages themselves do not contain kinematic loops, so this is no demand to the Modelica environment, but of course this would impose severe limitations to the vehicle models to use the tires.

Without support of the `MultiBody` package it is still possible to use the primary interface functions to the CTI functions. It could be possible to build some simple vehicle models by means of equations and to use FTire tires by directly calling these interface functions. For some special applications like longitudinal dynamics also an embedding of this basic interface into the `Modelica.Mechanics.Translational` package is conceivable.

- Ability to visualize surfaces like done in the `Visualizers` package. This means support of the undocumented "magic" functions `PackShape` and `PackMaterial` and the form numbers causing subsequent output variables having a meaning for animation (like used in earlier versions of the MSL, now moved to the Dymola version of the `ModelicaServices` package since MSL 3.1).

If a Modelica environment at least recognizes these functions (with some idle dummy implementation) the visualizers should not cause problems, but the models stay without tire or road visualization. If the surface visualizer model does not even compile it can be replaced by a dummy implementation easily.

9 Future Work Using upcoming Modelica Features

Since about version CTI 2.9 FTire provides the feature of using own tire or road models to incorporate into FTire [3]. The CTI subroutines for doing so contain pointers to functions in their argument lists. The currently used method of using C wrapper functions (cf.

section 4.2) has the drawback that the user subroutines have to be provided as C functions too.

With the upcoming feature of using functions as input argument to functions [8] it will be possible to write these user subroutines directly as Modelica functions and no C code will be needed any more.

10 Some Remarks on Modelica

But even with Modelica 3.2 there are still some issues and limitations for further development:

10.1 No Way of passing Information from Model to Function

Writing a *function* to evaluate the current height at a position on a road is appropriate for some special solid road geometry. But when it comes to write an elaborate soft soil road model in Modelica it is quite natural to use differential equations for doing so. This cannot be done in a function but only in a *model*. On the other hand a *function* has to be passed to the CTI function. Due to the requirement that Modelica functions are always pure (i.e. functions in the mathematical sense) there is no way of passing information from a model to a function. So even with Modelica 3.2 it will not be possible to use a user road model defined by a Modelica model in the CTI function for force calculation with user defined road.

10.2 Passing information for Co-Simulation

One aspect of the Modelica philosophy is that the user ought to build physically sound models in the Modelica equations, perhaps provide function derivations or inverses by means of annotations, but let the integrator do his business in solving these equations.

This is quite ok as long as pure Modelica is concerned. But this concept becomes questionable when there is a co-simulation: FTire does not simply calculate forces and torques based on the wheel center motion states as input quantities, but does its own integration (of its own internal model of the tire as a system of elements connected by springs forming the flexible ring).

When FTire is called from a variable step size integrator it will happen that an integration step fails and it will be repeated with some smaller time step. This means time goes back for the force calculation function. The FTire force calculation functions provide an input `MODE` for the "job control", i.e. here you can

inform FTire if the wheel input states are already accepted by the calling integrator or if they are not yet accepted. Currently we are only left with the option to call the function regardless if the the states are accepted or not. The FTire package works well with fixed step size integrators (like it is recommend for FTire usage anyway). But it can be expected that the performance with variable step size integrators could be improved if there were any means to communicate such information from the calling integrator to the FTire integrator somehow.

10.3 Multithreading

As far as we know currently no Modelica environment supports multi-threading, i.e. the parallel execution of parts of code and there are no means in Modelica to organize such parallelization. As long as just pure Modelica is concerned it might be argued that is a matter of the Modelica environment to see if parallelization is possible based on the equations (possibly in the flattened model). At least regarding external functions this approach seems to be not feasible: The environment needs to know if a function is thread save (can be parallelized without the risk of wrong results e.g. due to overwritten memory) and on the other hand if it is worth to parallelize the execution because it will take considerable time.

The latest versions of the CTI also provide versions of the force calculating functions for multi-core architectures. The forces of each tire instance are calculated by a separate thread. By this means the forces at all tires of a vehicle are calculated in parallel. The force calculation routine is split in two: There is one routine to pass the wheel center states and to trigger the force calculations in one separate thread for each instance. A second function fetches the results of the instances. In an *algorithmic* programming style for the simulation of the complete vehicle both functions can be used easily: In the algorithm for a single integrator iteration after the (preliminary) calculation of the wheel center states for all wheels there is first one loop triggering the force calculation for *all* tires and second another loop to fetch the results once the calculation is finished. In contrast in the Modelica setting, where external functions are called in some equation based model it seems likely that the force fetching function for the first instance is called before the calculation triggering function for the other instances, i.e. in fact there is no parallelization. This question needs further investigation. Probably there will arise the need to tell the Modelica environment (by way of some an-

notation) that such a pair of external function belongs together, that the second one delivers the results of the calculations triggered by the first one.³

10.4 Automatic inner components

The common properties model also contains the road visualizer as a sub-component. It can be turned on and off by some Boolean parameter. Using this instance of the road visualizer is sensible in all circumstances except for the rare case that the road is not fixed to the ground but to some other part (i.e. a RoadPart component is used). Considering this the default value for the road animation ought to be true.

But it is a (generally nice) feature of the Modelica inner / outer mechanism that it is *not* required to define an inner component explicitly in a top level model using components referencing on this model as an outer component: An implicit component is used in such a case. In the case of the common properties model here a component would have to be defined explicitly just to turn off road animation when it is not desired. To prevent this the default value for the road animation is false.

What were useful here is some function to determine *in* an inner model if an instance of it is the automatically created or an explicit one. With this feature it would be possible to set the value of the road animation parameter to true only if the component is explicitly defined.

11 Conclusions

The FTire package makes the FTire tire model available for the Modelica world. There are even some features supported like dynamic tire and road shape animation not yet available at the embedding in other multi-body programs. On the other hand there are many issues for further improvements like supporting custom road and tire models in a convenient way or parallelization of the tire force calculation for several tires. The usability in other Modelica environments has to be tested.

The original plan to develop also interfaces to other tire models was not addressed anymore within the Euroslib project due to lack of time but are subject to further work. Although any interface to the FTire tire

³Recently there have been considerations on co-simulation in a general frame in the Modelisar project [9]. It is not yet clear if the current CTI implementation is compatible to this FMI approach and in case it is, if this method of co-simulation is appropriate to interface a set of external function to Modelica.

model does not fit into the decomposition framework of a contact point based tire model [1] some other common framework for interfacing to other tire model is imaginable: Any tire model providing some API will have comparable tasks like reading data files or doing the force calculation. So above the CTI specific interface level a new tire model independent interface level may be established branching to the CTI routines or the counter-parts of some other tire model. In this case it ought to be possible to implement the embedding into the MultiBody framework independently from the FTire model but only accessing the tire model intermediate interface level.

12 Acknowledgements

The FTire (FTire interface for Dymola), FTireVDL (FTire tires in the VehicleDynamics package framework) and the FTireSimVis (FTire tires and roads visualized with the SimVis program) packages and the used geometry visualization package Visualizers have been developed as part of the ITEA2 **Eurosyslib** project (WP 8.5).

Earlier versions of the FTire interface package have been intensively tested in a diploma thesis [10] on a model of Kämmerer's side car vehicle "mython" [11] and at Dassault Aviation [12] with aircraft models. The package benefitted much from the reported bugs and suggestions for improvements.

References

- [1] ANDRES, Markus, ZIMMER, Dirk and CELLIER, François E.: Object-Oriented Decomposition of Tire Characteristics Based on Semi-Empirical Models. Proceedings of the 7th Modelica Conference, Como, Italy, 2009
- [2] The FTire homepage: www.cosin.eu/prod_FTire
- [3] The CTI reference document: www.cosin.eu/res/cti.pdf
- [4] The Visualization package product page: www.bausch-gall.de/vi1.htm
- [5] The VehicleDynamics package product flyer: www.modelon.se/DATAUPLOAD/File/Flyer_dymola_VDL_Car.pdf
- [6] DRENTH, Edo, GÄFVERT, Magnus: Modelica Delft-Tyre Interface. Proceedings of the 8th Modelica Conference, Dresden, Germany, 2011
- [7] BELLMANN, Tobias: Interactive Simulations and Advanced Visualizazion with Modelica. Proceedings of the 7th Modelica Conference, Como, Italy, 2009
- [8] Modelica 3.2 Language Specification, 12.4.2, Modelica Association, March 2010, www.modelica.org/documents/ModelicaSpec32.pdf
- [9] Functional Mock-Up Interface for Co-Simulation. Modelisar (07006). Document version 1.0, October 12th, 2010
- [10] ZAPF, Stefan: Aufbau und Validierung des Gesamtfahrzeug-Mehrkörpersimulationsmodells mit einem Hochfrequenzreifenmodell im Programmsystem Dymola, diploma thesis, Amberg, Germany, 2009.
- [11] The "mython" at Kämmerer's homepage: www.kaemmerer-group.com/mython/
- [12] THOMAS, Eric and LAPEYRE, Arnaud: DTG 121069 FTire Model-Evaluation Report (*unpublished*), 2010
- [13] www.itea2.org/public/project_leaflets/EUROSYSLIB_profile_oct-07.pdf

Implementation of the Spur Involute Gear Model on Modelica

Ivan Kosenko Il'ya Gusev

Russian State University of Tourism and Service, Department of Engineering Mechanics
Cherkizovo-1, Moscow region, 141221, Russia

Abstract

A procedure to build up a dynamical model of the gearbox with spur involute mesh is being described. The main attention is paid to the design technology of the cylindrical bodies elastic contact models. To track geometry of contact implicit equations of algebraic/transcendental or differential-algebraic type are being used. At the same time dynamical models of the bodies involved, gearwheels and gearbox housing, continue to be three-dimensional. Analytical computational procedures to obtain gradients and Hessians are constructed for implementing the contact tracking algorithm for the involute guided cylindrical surfaces. The known Johnson model is applied for computing the contact elastic normal force. This force is defined as an implicit function of the mutual penetration depth at contact. Regular algorithm to compute the normal elastic force is built up. This algorithm is proved to be convergent. A detailed analysis of the virtual setup dynamic model is carried out.

Keywords: spur gear; involute; Johnson model; mesh properties; tracking algorithm

1 Introduction

Computer modeling and simulation of dynamics for gearboxes of different kinds is a wide spread engineering task. One might highlight here two extreme poles of approaches for models constructing. Firstly, the finite element method can be used for building up sufficiently detailed dynamical models. It is clear that the models created using such an approach consume quite significant amount of computational resources. Secondly, on the other pole of models range one might find simplified models of gearboxes dynamics allowing a very fast models for machines and their units to develop. Examples of such models are presented, for instance, in the Modelica Standard Library. In addition, there exist well developed models taking into account friction forces during the mesh processes in

gearboxes [1]. One might also find several other interesting examples of the machinery applications including gearboxes models on Modelica [2, 3].

It is important to us to consider models incorporating both the rigid body dynamics sufficiently effective from the computational viewpoint and more detailed mesh models with different types of compliant contacts between teeth of gearwheels. The simplest problem in this way is the spur involute gear model implementation.

2 Cylindrical symmetry of 3D-bodies contact

Staying in frame of the spatial multibody dynamics classes previously developed [4, 5] it is quite natural to use additional rigid body C playing the role of platform, for implementing a relative planar motion of the bodies, two gearwheels denoted as A and B in our case. These bodies assumed to have cylindrical shapes and are able to move in the plane orthogonal to their generatrix. Let O_Cxyz be a coordinate system rigidly connected with the body C , and for definiteness let O_C be its center of mass. Assume the generatrix is always collinear to the axis O_Cz which can be expressed by the geometrical condition $\mathbf{k}_\alpha = \mathbf{k}_C$ ($\alpha = A, B$), where \mathbf{k}_α are the axis $O_\alpha z_\alpha$ unit vectors and \mathbf{k}_C is the unit vector of the body C axis O_Cz . To keep bodies' motion parallel to the coordinate plane O_Cxy one has to require two algebraic conditions for the bodies A and B mass centers z -coordinates: $z_{O_A} = \text{const}$, $z_{O_B} = \text{const}$ to be satisfied. All coordinates are assumed with respect to (w.r.t.) the frame O_Cxyz .

Algebraic equations mentioned can be easily implemented in implicit form if one uses, for instance, constraints of the joint type [4] to fix the bodies A and B in the body C . In this case, the body C itself can perform arbitrary spatial motions. We consider its movement as being convective in compound motions of the bodies A and B w.r.t. certain inertial frame of reference. Thus it is quite natural to call the body C as a gear

housing, and the bodies A and B are supposed to play the role of gearwheels.

One might build up mechanical tools for the cylindrical bodies contact using 2D-geometry techniques with aid of the above reduction to the plane O_Cxy . For instance, the cylinders contact tracking model might be written, similarly to [6], in the form of six, algebraic or transcendental, equations as follows

$$\begin{aligned} \text{grad } g_A(\mathbf{r}_{P_A}) &= \lambda \text{grad } g_B(\mathbf{r}_{P_B}), \\ \mathbf{r}_{P_A} - \mathbf{r}_{P_B} &= \mu \text{grad } g_B(\mathbf{r}_{P_B}), \\ g_A(\mathbf{r}_{P_A}) &= 0, \quad g_B(\mathbf{r}_{P_B}) = 0, \end{aligned} \quad (1)$$

where $\mathbf{r}_{P_\alpha} = (x_{P_\alpha}, y_{P_\alpha})^T$ ($\alpha = A, B$) are the radius vectors for the points P_A, P_B under tracking w.r.t. the housing coordinate system O_Cxy ; functions $g_\alpha(\mathbf{r})$ express equations for the curves bounding planar figures of the bodies A and B w.r.t. the axes O_Cxy ; λ, μ are the auxiliary scalar variables. Totally the system (1) has six scalar equations w.r.t. six scalar variables $x_{P_A}, y_{P_A}, x_{P_B}, y_{P_B}, \lambda, \mu$.

To complete classes corresponding to models of contact one has to define contacting curves in the bodies own planar coordinate systems $O_\alpha x_\alpha y_\alpha$ in the form $f_\alpha(x_\alpha, y_\alpha) = 0$. If T_α is an orthogonal 2×2 -matrix defining current orientation of the body α planar figure then obviously the relations $g_\alpha(\mathbf{r}) = f_\alpha[T_\alpha^T(\mathbf{r} - \mathbf{r}_{O_\alpha})]$, $\text{grad } g_\alpha(\mathbf{r}) = T_\alpha \text{grad } f_\alpha[T_\alpha^T(\mathbf{r} - \mathbf{r}_{O_\alpha})]$ ought to take place.

Similarly, following the paper [7] one might easily construct a model to track the cylindrical contact by introducing a system of differential-algebraic equations of the form

$$\begin{aligned} \dot{\mathbf{r}}_{P_A} = \mathbf{u}_{P_A}, \quad \dot{\mathbf{r}}_{P_B} = \mathbf{u}_{P_B}, \quad \dot{\lambda} = \xi, \quad \dot{\mu} = \eta, \quad (2) \\ [\boldsymbol{\omega}_A, \text{grad } g_A] + T_A \text{Hess } f_A T_A^T (\mathbf{u}_{P_A} - \mathbf{v}_{P_A}) - \\ \xi \text{grad } g_B - \\ \lambda ([\boldsymbol{\omega}_B, \text{grad } g_B] + T_B \text{Hess } f_B T_B^T (\mathbf{u}_{P_B} - \mathbf{v}_{P_B})) = \mathbf{0}, \\ \mathbf{u}_{P_A} - \mathbf{u}_{P_B} - \eta \text{grad } g_B - \\ \mu ([\boldsymbol{\omega}_B, \text{grad } g_B] + T_B \text{Hess } f_B T_B^T (\mathbf{u}_{P_B} - \mathbf{v}_{P_B})) = \mathbf{0}, \\ (\text{grad } g_A, \mathbf{u}_{P_A}) - (\text{grad } f_A, T_A^T \mathbf{v}_{P_A}) = 0, \\ (\text{grad } g_B, \mathbf{u}_{P_B}) - (\text{grad } f_B, T_B^T \mathbf{v}_{P_B}) = 0, \end{aligned} \quad (3)$$

where the vectors $\mathbf{v}_{P_A}, \mathbf{v}_{P_B}$ are relative, w. r. t. the body C , velocities of the bodies physical points currently located at the geometrical points P_A, P_B . One might calculate them according to the Euler formula

$$\mathbf{v}_{P_\alpha} = \mathbf{v}_{O_\alpha} + [\boldsymbol{\omega}_\alpha, \mathbf{r}_{P_\alpha} - \mathbf{r}_{O_\alpha}] \quad (\alpha = A, B),$$

where O_A, O_B are the bodies mass-centers mentioned above, $\boldsymbol{\omega}_A, \boldsymbol{\omega}_B$ are relative w. r. t. the housing angular velocities of the bodies, always directed along the

O_Cz -axis. Note that the points O_A, O_B might be located at different levels of the O_Cz -axis of the gearbox housing. But nevertheless one should regard the equations (2), (3) in their planar version as being projected onto the plane O_Cxy .

In this case, for complete implementation of the contact model, we need to compute the gradients $\text{grad } f_\alpha$ and Hessians $\text{Hess } f_\alpha$ at opposing points P_α in bodies own coordinates.

3 Geometry of the spur involute gear

One has to consider the involute equation in the plane $O_\alpha x_\alpha y_\alpha$ of the gearwheel coordinate system $O_\alpha x_\alpha y_\alpha z_\alpha$ for resolving the problem which has been formulated above for the case of the spur involute meshing. For this one has to apply polar coordinates R_α, θ_α defined for each body α in the following known way: $x_\alpha = R_\alpha \cos \theta_\alpha, y_\alpha = R_\alpha \sin \theta_\alpha$. For the involute unwinding counterclockwise an equation for the polar coordinates can be deduced from the known relations [8] in the form

$$\frac{\sqrt{R_\alpha^2 - r_{\alpha b}^2}}{r_{\alpha b}} - \arccos \frac{r_{\alpha b}}{R_\alpha} - \theta_\alpha = 0, \quad (4)$$

where $r_{\alpha b}$ is the involute base circle radius. To compute $\text{grad } f_\alpha$ and $\text{Hess } f_\alpha$ one has to use formulae of the transformation $(x_\alpha, y_\alpha) \mapsto (R_\alpha, \theta_\alpha)$ and apply an auxiliary Jacobi matrices arising in the process of analytical calculations.

The contact tracking algorithm developed requires equation of a curve in the form $f(x, y) = 0$ instead of equation (4). Introducing the notation

$$p(R, \theta) = \frac{\sqrt{R^2 - r_b^2}}{r_b} - \arccos \frac{r_b}{R} - \theta \quad (5)$$

one can see easily that

$$p(R, \theta) = f(R \cos \theta, R \sin \theta). \quad (6)$$

This equation is a starting point for producing all the formulae for gradients and Hessians. Indeed by virtue of (6) we have

$$\text{grad } p = (f_x, f_y) \begin{pmatrix} x_R & x_\theta \\ y_R & y_\theta \end{pmatrix} = \text{grad } f \frac{\partial(x, y)}{\partial(R, \theta)} \quad (7)$$

since $\text{grad } p = (p_R, p_\theta)$. Therefore

$$\text{grad } f = \text{grad } p \left[\frac{\partial(x, y)}{\partial(R, \theta)} \right]^{-1}. \quad (8)$$

One can see from (5) that

$$\text{grad } p = \left(\frac{\sqrt{R^2 - r_b^2}}{r_b R}, -1 \right), \quad (9)$$

and from the polar coordinates definition the relation

$$\frac{\partial(x, y)}{\partial(R, \theta)} = \begin{pmatrix} \cos \theta & -R \sin \theta \\ \sin \theta & R \cos \theta \end{pmatrix}.$$

follows. Hence,

$$\left[\frac{\partial(x, y)}{\partial(R, \theta)} \right]^{-1} = \begin{pmatrix} \cos \theta & \sin \theta \\ -\frac{\sin \theta}{R} & \frac{\cos \theta}{R} \end{pmatrix}. \quad (10)$$

To complete the computation of $\text{grad } f$ as a function of x, y one can remark that

$$R = \sqrt{x^2 + y^2}, \quad \cos \theta = \frac{x}{R(x, y)}, \quad \sin \theta = \frac{y}{R(x, y)}. \quad (11)$$

From (7) for the Hessian computation note that

$$\text{Hess } p = \begin{pmatrix} \frac{\partial(x, y)}{\partial(R, \theta)} \end{pmatrix}^T \text{Hess } f \frac{\partial(x, y)}{\partial(R, \theta)} + f_x \text{Hess } x + f_y \text{Hess } y, \quad (12)$$

where the following notation has been used

$$\text{Hess } p = \begin{pmatrix} p_{RR} & p_{R\theta} \\ p_{\theta R} & p_{\theta\theta} \end{pmatrix} = \begin{pmatrix} \frac{r_b}{R^2 \sqrt{R^2 - r_b^2}} & 0 \\ 0 & 0 \end{pmatrix},$$

$$\text{Hess } x = \begin{pmatrix} x_{RR} & x_{R\theta} \\ x_{\theta R} & x_{\theta\theta} \end{pmatrix} = \begin{pmatrix} 0 & -\sin \theta \\ -\sin \theta & -R \cos \theta \end{pmatrix},$$

$$\text{Hess } y = \begin{pmatrix} y_{RR} & y_{R\theta} \\ y_{\theta R} & y_{\theta\theta} \end{pmatrix} = \begin{pmatrix} 0 & \cos \theta \\ \cos \theta & -R \sin \theta \end{pmatrix},$$

Now one can obtain from equation (12) a formula for the Hessian we sought

$$\text{Hess } f = \left[\left(\frac{\partial(x, y)}{\partial(R, \theta)} \right)^T \right]^{-1} \times (\text{Hess } p - f_x \text{Hess } x - f_y \text{Hess } y) \left(\frac{\partial(x, y)}{\partial(R, \theta)} \right)^{-1}.$$

All the objects included here on the right hand side have explicit expressions in polar coordinates. It is evident, these expressions can be resolved w. r. t. coordinates x, y with aid of transformation (11).

4 Contact force model

The geometrical properties have been implemented as a class parameter for the base template described in [5] for the model outlined above with contact of involutes. Along with the geometrical properties model, the contact elastic normal force model plays a key role as well. According to the results of [9] the Johnson contact model [10] seems to be the most acceptable one for the case of the cylindrical bodies contact. The model can be expressed as an equation written for the case of so-called external contact

$$h = \frac{N}{\pi E^*} \left[\ln \frac{4\pi E^* (\rho_A + \rho_B)}{N} - 1 \right], \quad (13)$$

representing an implicit function $N(h)$ for the specific normal elastic force, per unit of length along the cylinder generatrix, depending on the depth h of mutual approach (penetration). Here in equation (13) E^* is the composite modulus of elasticity for the contact. It satisfies the equation

$$\frac{1}{E^*} = \frac{1 - \nu_A^2}{E_A} + \frac{1 - \nu_B^2}{E_B},$$

where E_A, E_B are Young's moduli of bodies' material, ν_A, ν_B are Poisson ratios. Values ρ_A, ρ_B are radii of curvature for involutes in the mesh each computed at current positions of the points P_A, P_B respectively.

Remark 1 *Staying in frame of the Hertz model conditions, we assume the contact area dimensions small as compared with the sizes of contacting bodies. Thus, the cylindrical involute surfaces in vicinities of points P_A, P_B are approximated by the circular cylinders with an accuracy of order higher than two. Application of equation (13) means that we virtually replace the cylindrical surfaces with involutes as guides by the circular cylinders with the same radii of curvature at any current instant of simulation time. Evidently these cylindrical surfaces, involutive and circular ones, have mutual tangency of the second order.*

Computational implementation of formula (13) inversion reduces to an equation w.r.t. dimensionless variables x, y defined in the following way:

$$x = \frac{e}{4(\rho_A + \rho_B)} h, \quad y = \frac{e}{4\pi E^* (\rho_A + \rho_B)} N.$$

Then (13) becomes equivalent to the equation

$$y \cdot \ln y = -x \quad (14)$$

defining the implicit function $y(x)$.

To resolve equation (14) correctly in vicinity of its evident solution $x = 0$, $y = 0$ one has to find a segment of monotonicity of the left hand side of (14). The zero solution corresponds to the case of beginning of the contacting process. In this case, a contact patch, generically rectangular area, degenerates into the 1D-line segment along the cylindrical surfaces generatrix. It is easy to see that the segment sought is $[0, e^{-1}]$. Within this segment the left hand side function of (14) decreases monotonically from zero to $-e^{-1}$. Thus one can define an area of applicability of the Johnson model by the following inequalities

$$h \leq \frac{4(\rho_A + \rho_B)}{e^2}, \quad N \leq \frac{4\pi E^*(\rho_A + \rho_B)}{e^2}. \quad (15)$$

From the involute properties and using Figure 1 the relation $\rho_A + \rho_B = |K_A K_B|$ is satisfied to within the (small enough) value h . For this reason, if the material stiffness is sufficiently large then the depth h is small enough, and then the left condition of (15) is always satisfied. Here, the points K_A , K_B are the points of a tangency between the line of action and base circles of gearwheels.

For real materials even the condition $h \ll |K_A K_B|$ is satisfied. Then for Young's moduli large enough the Johnson model is surely valid on the segment $y \in [0, e^{-1}]$ of monotonicity for the left side of equation (14). Since the derivative for the left side of equation (14) at $y = e^{-1}$ is equal to zero then we may furthermore restrict ourselves by set of strict monotonicity corresponding to the condition $0 \leq x < e^{-1}$, or equivalently to $0 \leq y < e^{-1}$.

Moreover, equation (14) has a singularity at zero. Therefore, we shall construct an algorithm for computing the function $y(x)$ taking into account that $0 < x < e^{-1}$. To proceed with the algorithm replace an unknown function $y(x)$ in equation (14) by the function $\eta(x)$ according to the formula $y(x) = x\eta(x)$. Then a new equation has the form

$$\eta(\ln x + \ln \eta) + 1 = 0. \quad (16)$$

Let us introduce here a new known independent variable μ instead of the old one x according to the equation

$$v = -\frac{1}{\ln x}.$$

The value v is small and positive if the value x is small and positive. Then equation (16) is transformed to the form

$$\eta = v(1 + \eta \ln \eta) \quad (17)$$

more suitable for investigating and computing the solution for the given value of v .

To overcome the problem we need in an algorithm with the behavior regular enough in vicinity of zero. It turned out equation (17) delivers also an iteration process in the explicit form

$$\eta_{n+1} = v(1 + \eta_n \ln \eta_n), \quad n = 0, 1, \dots \quad (18)$$

One might set any value η_0 satisfying the condition $0 < \eta_0 \leq e^{-1}$ as a guess value of iteration process (18). It is easy to see the numeric sequence $\{\eta_n\}_{n=0}^{\infty}$ built up using process (18) is strictly positive and bounded: $0 < \eta_n < v$. Therefore, this sequence has at least one limit point η_* . It is equivalent to an existence of the subsequence $\{\eta_{n_k}\}_{k=0}^{\infty}$ converging to this limit $\eta_{n_k} \rightarrow \eta_*$ as $k \rightarrow \infty$.

This limit is unique. Indeed, by virtue of (18) the limit satisfies equation (17). If there would be another different limit η_{**} then equation (17) should have at least two different solutions on the set $[0, 1)$. Then as a consequence equation (14) should have two different solutions on the set $[0, e^{-1})$ what is impossible because it has exactly one solution on this set.

Computations show that iteration process (18) converges fast enough. Merit of the process is that it works equally well for all admissible values of x . If x becomes close to zero then the value $v > 0$ is also small. Besides, for any arbitrarily small $\eta > 0$ the function $\eta \ln \eta$ always stays uniformly bounded. Thus the iteration operator conserves its regularity for any admissible x .

Other class parameters of the contact model template in our case are following: (a) normal viscous term was selected similar to the implementation described in [7]; (b) tangent friction force model for definiteness and simplicity, similar, for example, to the paper [11], was selected as a regularized Coulomb friction law having a shape of the piecewise linear function of relative velocity at contact [6]. There are no difficulties for changing the corresponding class parameter and for applying any different model, more complicated than ones mentioned above.

5 Algorithm of teeth pairs switching

To describe the algorithm we assume gearwheels in rotational motion each such that the pinion, wheel A , rotates clockwise, and the gear (B) does it counter-clockwise. We consider a process of initial data generation later. When boosting the pinion tooth in generic case starts to penetrate the corresponding gear tooth

$|\text{grad } g_A|$, $|\text{grad } g_B|$ (co)vectors and then use the first and the second equations of the system (1). Gradients are to be computed at points P'_A and P'_B respectively. Since we always suppose that gradients are directed “outside” the bodies at contact, we have to assume that

$$\lambda(t_*) < 0, \quad \mu(t_*) < 0.$$

Furthermore, since functions g_A , g_B are derived from the functions f_A , f_B by translatory and rotary motions of the three dimensional space \mathbf{R}^3 then the following condition is satisfied

$$|\text{grad } g_\alpha| = |\text{grad } f_\alpha| \quad (\alpha = A, B).$$

To compute the value $|\text{grad } f_\alpha|$ let us apply equations (8), (9), (10) from above in the following way

$$\begin{aligned} \text{grad } f_\alpha &= \left(\frac{\sqrt{R_\alpha^2 - r_{\alpha b}^2}}{r_{\alpha b} R_\alpha}, -1 \right) \begin{pmatrix} \cos \theta_\alpha & \sin \theta_\alpha \\ -\frac{\sin \theta_\alpha}{R_\alpha} & \frac{\cos \theta_\alpha}{R_\alpha} \end{pmatrix} \\ &= \left(\frac{\sqrt{R_\alpha^2 - r_{\alpha b}^2}}{r_{\alpha b} R_\alpha}, -\frac{1}{R_\alpha} \right) \begin{pmatrix} \cos \theta_\alpha & \sin \theta_\alpha \\ -\sin \theta_\alpha & \cos \theta_\alpha \end{pmatrix}, \end{aligned}$$

where R_α , θ_α are the polar coordinates in the body α coordinate system $O_\alpha x_\alpha y_\alpha$. One can see easily from last equation that the (co)vector $\text{grad } f_\alpha$ is a result of the (co)vector

$$\left(\frac{\sqrt{R_\alpha^2 - r_{\alpha b}^2}}{r_{\alpha b} R_\alpha}, -\frac{1}{R_\alpha} \right)$$

rotation by the angle θ_α . Then one can write down the norm sought as

$$|\text{grad } g_\alpha| = |\text{grad } f_\alpha| = \frac{1}{r_{\alpha b}} \quad (\alpha = A, B).$$

Thus the gradient (co)vector of the function defining the involute has a constant norm inversely proportional to the base circle radius. Now from the first and the second equations of the system (1) we have respectively

$$\begin{aligned} \lambda(t_*) &= -\frac{|\text{grad } g_A|}{|\text{grad } g_B|} = -\frac{r_{Bb}}{r_{Ab}}, \\ \mu(t_*) &= -\frac{|\mathbf{r}_{P_B}(t_*) - \mathbf{r}_{P_A}(t_*)|}{|\text{grad } g_B|} = \\ &= -\frac{r_{Bb} |\mathbf{r}_{P_B}(t_*) - \mathbf{r}_{P_A}(t_*)|}{r_{Ab} |\mathbf{r}_{P_B}(t_*) - \mathbf{r}_{P_A}(t_*)|}. \end{aligned}$$

And useful in all aspects result was obtained by the way:

Assertion 2 *In case of involutes the equation*

$$\lambda = -\frac{r_{Bb}}{r_{Ab}}$$

represents an integral of motion for the contact tracking system of DAEs (2), (3).

Indeed, one can see from above that the variable λ keeps its initial value all time of simulation

$$\lambda(t) \equiv \lambda(t_0) = \lambda(t_*) = -\frac{r_{Bb}}{r_{Ab}}.$$

This property may be very useful to control an accuracy of computations during the simulation process.

6 Some details of implementation

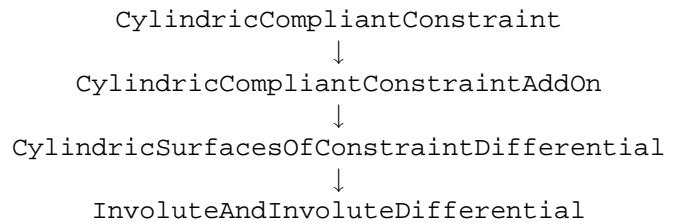
As it was already mentioned the template

`ContactConstraintBaseTemplate`

developed earlier [5] has been applied for implementing the contact model of two gearwheels with spur involute gearing. The template has four class parameters defining models of: (a) the normal elastic force, (b) the normal force of viscous resistance, (c) the tangent force of resistance for relative slipping at contact, (d) geometry of surfaces in vicinity of contact, in our case they are the cylinders guided by an involute.

As it was already noted we guess approximately that a contact patch is the rectangular strip, in general thin enough. The normal elastic force is assumed to be uniformly distributed in one dimension along the generatrix of cylinder over the patch.

Let us consider in more details an implementation of the class parameter responsible for geometry properties of the contact. This class is implemented as a four-level hierarchy of inheritance for the properties and behavior:



A differential-algebraic equations are applied here for implementing the contact tracking algorithm.

A base class for all geometry classes implementing cylindrical contact is the model `CylindricCompliantConstraint`. This class is responsible for computation of geometric and kinematic properties of the points P_A and P_B under tracking w. r. t. the third body C , the gearbox housing. These properties are described in particular by the following variables:

- r_{Ar} , r_{Br} are the points P_A and P_B relative positions;
- $gradg_{Ar}$, $gradg_{Br}$ are the cylindrical surfaces gradient vectors computed at points P_A , P_B respectively in the frame of coordinates O_{Cxyz} ;
- $norm_{Ar}$ is the unit vector normal to an outer surface of the body A computed at the point P_A w. r. t. the coordinate system O_{Cxyz} ;
- vr_{Ar} , vr_{Br} are relative velocities of bodies' A , B points currently occupying the geometric points P_A , P_B locations;

Modelica code of the current class under description, its section of equations, reads

equation

```

rA = InPortC.r + InPortC.T*rAr;
rB = InPortC.r + InPortC.T*rBr;
gradgA = InPortA.T*gradfA;
gradgB = InPortB.T*gradfB;
normA = gradgA/sqrt(gradgA*gradgA);
gradgAr = (transpose(InPortC.T)*
  InPortA.T)*gradfA;
gradgBr = (transpose(InPortC.T)*
  InPortB.T)*gradfB;
normAr =
  gradgAr/sqrt(gradgAr*gradgAr);
vrA = InPortA.v + cross(InPortA.omega,
  rA - InPortA.r);
vrB = InPortB.v + cross(InPortB.omega,
  rB - InPortB.r);
vrAe = InPortC.v +
  cross(InPortC.omega,
  rA - InPortC.r);
vrBe = InPortC.v +
  cross(InPortC.omega,
  rB - InPortC.r);
vrAr = transpose(InPortC.T)*
  (vrA - vrAe);
vrBr = transpose(InPortC.T)*
  (vrB - vrBe);

```

end CylindricCompliantConstraint;

In the derived class CylindricCompliantConstraintAddOn:

- PA , PB are variables for coordinates of the points where a resultant contact forces are applied, in directions of bodies A and B respectively;
- $relvnr$ is the normal component, in case of $PA = PB$, of the velocity for the point PA of the body A relative to the body B ;
- $relvtr$ is the tangent component of the the body A relative velocity at $PA = PB$ w. r. t. the coordinate system O_{Cxyz} ;

- $kappa$ is the contact indicator which is: (a) equal to zero for the case of the surfaces touching each other by segment of strait line, (b) positive and equals to the distance between the surfaces for the case of contact absence, (c) negative and characterizes depth of mutual penetration for the case of contact, contact patch has a rectangular shape;
- $kappaA$, $kappaB$ are the parameters defining elastic properties of bodies A and B respectively.

Section of equations/behavior for this model has the following code

equation

```

kappa = sqrt(gradgBr*gradgBr)*mu;
if noEvent(kappa <= 0) then
  PA = kappaA*rB + kappaB*rA;
  PB = PA;
else
  PA = rA;
  PB = rB;
end if;
vPA = transpose(InPortC.T)*
  (InPortA.v + cross(InPortA.omega,
  PA - InPortA.r));
vPB = transpose(InPortC.T)*
  (InPortB.v + cross(InPortB.omega,
  PB - InPortB.r));
relv = vPA - vPB;
relvnr = relv*normAr;
relvn = InPortC.T*relv*normAr;
vPA_n = vPA*normAr;
vPB_n = vPB*normAr;
vPA_t = vPA - vPA_n*normAr;
vPB_t = vPB - vPB_n*normAr;
relvtr = vPA_t - vPB_t;
relvt = InPortC.T*(vPA_t - vPB_t);
relvtsqrt = sqrt(relvt*relvt);
OutPortA.F = Forcet +
  Forcen*normA + Forcev*normA;
OutPortA.P = PA;
OutPortB.P = PB;

```

end CylindricCompliantConstraintAddOn;

The next derived class CylindricSurfacesOfConstraintDifferential implements the DAE system (2), (3). Its Modelica code is similar to one of the class SurfacesOfConstraintDifferential described in [7] for the generic case of the Hertz-point model. The difference concerns an account of the cylindrical symmetry for the current case.

Finally, the contact surfaces, rather curves bounding planar figures of bodies, specifications are defined in the last class of the inheritance chain InvoluteAndInvoluteDifferential. This model implements an algorithm described in Section 3 for computing of

gradients and Hessians in bodies' coordinates systems. As one can see from Section 3 corresponding Modelica code has to be bulky enough.

It is clear that when working two teeth of gearwheels A and B cannot stay in the meshing process during long time. One can see in Figure 1 that in case of the pinion rotation clockwise the contact point (segment of line), or rather contact patch small enough, moves from the point a to the point b of the line of action $K_A K_B$. At the very moment of contact loss for the current pair of teeth at the point b the next pair arrives at contact, the point a , and new "point" of contact starts its motion along the meshing straight line of action.

One has to note here that for simplicity we consider a mesh process without overlapping of time intervals for the teeth pairs contacting. If they are overlapped then one should create at least two contact objects "connecting" the objects of bodies A and B . These objects are to be activated/deactivated alternatively when arriving at/departing from the point a/b .

While the contact patch moves from its position a to the position b each of gearwheels A , B rotates by the pitch angle $\Delta\gamma_A$, $\Delta\gamma_B$ respectively. The last class of the inheritance line considered above besides the computation of gradients and Hessians implements also a switching process for the pairs of gearwheels thus synchronizing this switching with the corresponding angles of rotation for the bodies A and B .

This mechanism for discontinuous jumps of the contact points is implemented by the Modelica event handling facility. Code of the class `InvoluteAndInvoluteDifferential` fragment concerning required switching reads

```

...
der(phirel_A) = Active*omegarel_A[3];
der(phirel_B) = Active*omegarel_B[3];
Deltar = rBr - rAr;
when abs(phirel_A) > gamma_A +
    gamma_Astep then
    reinit(gamma_B, abs(phirel_B));
    reinit(rAr[1], rA0[1]);
    reinit(rAr[2], rA0[2]);
    reinit(rBr[1], rA0[1] + Deltar[1]);
    reinit(rBr[2], rB0[2] + Deltar[2]);
    reinit(lambda, lambda0);
    reinit(gamma_A, gamma_A +
        gamma_Astep);
end when;
...
    
```

Here the variables `phirel_A`, `phirel_B` are to accumulate an angles of rotation for the bodies A , B w. r. t. the housing C . They are the model state

variables having derivatives defined as z-components of the wheels relative angular velocities vectors `omegarel_A` and `omegarel_B`.

In the code fragment above, variables `rA0`, `rB0`, in addition to ones already described, correspond to the points P_A , P_B initial position, at the point a in Figure 1, vectors in the contact tracking algorithm. Relative, w. r. t. P_A , position of the point P_B for the new teeth pair being directed, as we know, along the line of action $K_A K_B$, has to be equal, because of rigidity, to the similar position for the previous pair losing contact at the event. This relative position is tracked by the variable `Delta`.

The variables `mu`, `lambda` correspond to the variables μ , λ in equations (2), (3); `lambda0` correspond to the λ initial value at the position a ; `gamma_A`, `gamma_B` are the variables for the gearwheels angles of rotation changing by the pitch values $\Delta\gamma_A$, $\Delta\gamma_B$ being stored in variables `gamma_Astep`, `gamma_Bstep`. Note that for correct handling of the switching process it is sufficient to track only the pinion, body A , angle of rotation and use only the variables `gamma_A`, `gamma_Astep`, `phirel_A`, `omegarel_A`. Initial depth of penetration, just after the contact switch, for the new pair is defined by the variable `mu` remains the same as for the previous pair of teeth in contact. This is because the gradient, from the right hand side of the second equation in system (1), norm stays constant in case of the involute. This constant is equal to the value $1/r_{Bb}$. For definiteness the wheel A supposed to rotate monotonically clockwise.

7 Computational experiments

To perform a computational testing program for the gearbox model one builds up a virtual setup consisting of two gearwheels: the pinion A and the driven gear B . For simplicity one assumes the gearbox housing C be fixed w. r. t. inertial frame of reference, and the origin O_C of its coordinate system O_Cxyz coincides with the pinion geometrical center O_A . Cylindrical revolute joint connecting the bodies A and C is also located at the point O_C . The gearwheel B center O_B is located on the horizontal axis O_Cx . Here, at O_B , a cylindrical revolute joint connecting the body B and the auxiliary slider S is located. The slider S is in turn able to slip freely w. r. t. the body C along the axis O_Cx , though this sliding performs with a resistance of the spring of high stiffness with the damper. This spring connects bodies C and S between one another.

We introduced in the current experimental setup un-

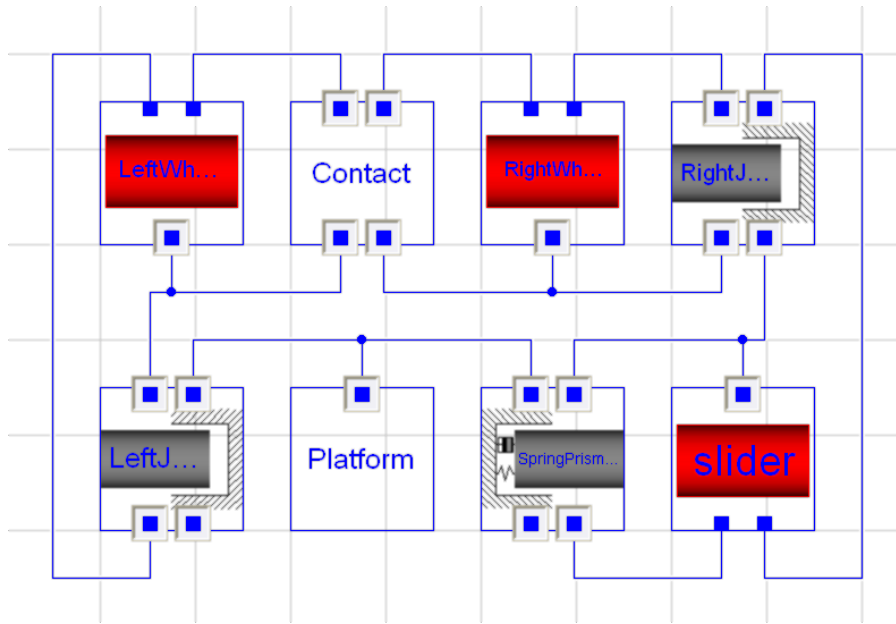


Figure 2: The Virtual Setup Visual Model

der description a compliance between the bodies B and C . This compliance is implemented by the auxiliary slider mentioned and is directed along the line $O_A O_B$ connecting the wheels centers and lying on the axis $O_C x$. Such a construct prevents the static indefiniteness in the model for the case of rigid contact at the mesh point of gearwheels A and B . A visual model of the setup is shown in Figure 2.

7.1 Parameters of the model

The following independent parameters are defined in the mesh model:

- $z_A = 20$ is number of teeth for the pinion;
- $z_B = 30$ is number of teeth for the driven gearwheel;
- $r_A = 0.2\text{m}$ is the pinion pitch circle radius.

After that the remaining geometry parameters of the mesh are computed as follows:

- $n = z_B/z_A$ is the transmission ratio;
- $r_B = nr_A$ is the pitch circle radius of the driven gearwheel;
- $\Delta\gamma_A = 2\pi/z_A$, $\Delta\gamma_B = 2\pi/z_B$ are the gearwheels angular pitches;

To define the mesh further it is essential to set the pressure angle α_w value. It can be chosen using the condition

$$\alpha_w > \alpha_{w\text{inf}},$$

where $\alpha_{w\text{inf}} = \inf \alpha_w$ is the lower bound for all the pressure angles permissible by the parameters above. Its value is defined by the formula

$$\alpha_{w\text{inf}} = \arctan \frac{2\pi}{z_A(1+n)}.$$

For definiteness we will use the following value

$$\alpha_w = 1.05\alpha_{w\text{inf}}.$$

Furthermore, using the pressure angle and transmission ratio one can compute sequentially all geometrical parameters needed which are shown in Figure 1. First the base circles radii can be found as

$$r_{\alpha b} = r_\alpha \cos \alpha_w \quad (\alpha = A, B).$$

Then one can compute a full length of the line of action as follows

$$|\overrightarrow{K_A K_B}| = r_A(1+n) \sin \alpha_w.$$

At the same time a length of any segment $[a, b]$ along this line is exactly the length of arc for any of the base circles corresponding to the pitch angle $\Delta\gamma_A$ or $\Delta\gamma_B$

$$|\overrightarrow{ab}| = r_\alpha \Delta\gamma_\alpha \quad (\alpha = A, B).$$

Initial distance between the gearwheels centers is equal to the value $L = r_A + r_B$. To compute initial conditions for the contact tracking system of DAEs (2), (3) we need in additional computations. From the description above we have for absolute initial coordinates of the points O_C and O_A

$$\mathbf{r}_{O_C} = \mathbf{r}_{O_A} = (0, 0, 0)^T.$$

Thus an initial position of the gear center is defined by the equation

$$\mathbf{r}_{O_B} = (L, 0, 0)^T.$$

Initial positions of the points K_A and K_B can be computed easily, see Figure 1, with use of the following vector formulae

$$\begin{aligned} \mathbf{r}_{K_A} &= \mathbf{r}_{O_A} + r_{Ab} (\cos \alpha_w, \sin \alpha_w, 0)^T, \\ \mathbf{r}_{K_B} &= \mathbf{r}_{O_B} - r_{Bb} (\cos \alpha_w, \sin \alpha_w, 0)^T. \end{aligned}$$

After that the line of action directing vector can be obtained as $\overrightarrow{K_A K_B} = \mathbf{r}_{K_B} - \mathbf{r}_{K_A}$. And now one can define a position of the point a , where the contact process begins, in the form

$$\mathbf{r}_a = \mathbf{r}_{K_A} + \frac{1}{2} \left(|\overrightarrow{K_A K_B}| - |ab| \right) \frac{\overrightarrow{K_A K_B}}{|\overrightarrow{K_A K_B}|},$$

and also an initial position of the point b , where the contact losses, has the representation

$$\mathbf{r}_b = \mathbf{r}_a + \frac{|ab|}{|\overrightarrow{K_A K_B}|} \overrightarrow{K_A K_B}.$$

After the endpoints a , b of an active segment of the line of action have been defined it is time to compute the addendum radius r_{Ba} of the gear as a distance between the point a and the initial position of O_B , see Figure 1. The addendum radius r_{Aa} of the pinion is in turn a distance between an initial position of the point b and O_A . These radii are defined by the equations

$$r_{Aa} = |\mathbf{r}_b - \mathbf{r}_{O_A}|, \quad r_{Ba} = |\mathbf{r}_a - \mathbf{r}_{O_B}|.$$

To compute initial angles of rotation for the pinion and gear we assume that at an initial instant of simulation teeth of an initial pair are touching each other geometrically without any pressure, and, as a result, mutual penetration is absent. An initial angular velocities of the gearwheels assumed equal to zero. For definiteness we also assume that the axis $O_A x_A$ of the body A crosses the base circle exactly at a root point of the involute. This involute defines a surface of the tooth contacting with its mate exactly at the point a . Similarly, the body B axis $O_B x_B$ passes through the root point of the contact involute of the body B at initial instant.

One can compute the polar angles of each the involute mentioned above using the equations (4) with the following equations ($\alpha = A, B$)

$$\theta_\alpha = \frac{\sqrt{|\mathbf{r}_a - \mathbf{r}_{O_\alpha}|^2 - r_{\alpha b}^2}}{r_{\alpha b}} - \arccos \frac{r_{\alpha b}}{|\mathbf{r}_a - \mathbf{r}_{O_\alpha}|},$$

Thus we can define an initial values for the angles of rotation of bodies A and B in the form

$$\varphi_\alpha(t_0) = \arg \zeta_\alpha - \theta_\alpha \quad (\alpha = A, B),$$

where complex numbers ζ_A, ζ_B are defined via the vectors $\mathbf{r}_a - \mathbf{r}_{O_\alpha}$ components as

$$\zeta_\alpha = (x_a - x_{O_\alpha}) + i(y_a - y_{O_\alpha}) \quad (\alpha = A, B).$$

Note that the function \arg of complex argument has a computer implementation as a standard library function `atan2`.

Initial quaternions of bodies A and B orientation are defined using known formulae

$$\mathbf{q}_\alpha(t_0) = \left(\cos \frac{\varphi_\alpha(t_0)}{2}, 0, 0, \sin \frac{\varphi_\alpha(t_0)}{2} \right)^T \quad (\alpha = A, B).$$

State variables of the DAE system (2), (3) tracking contact are to satisfy the following initial conditions

$$\mathbf{r}_{P_A}(t_0) = \mathbf{r}_{P_B}(t_0) = \mathbf{r}_a, \quad \mu(t_0) = 0, \quad \lambda(t_0) = -\frac{r_{Bb}}{r_{Ab}}.$$

Note here that in case of the involute the state variable $\lambda(t)$ turned out to be constant value

$$\lambda(t) \equiv \text{const} = \lambda(t_0).$$

Thus this equation represents exactly integral of motion, and it can be used effectively to control an accuracy of computations.

Finally, in the example under consideration the constant driving torque $M_A = (0, 0, -1 \text{ N} \cdot \text{m})^T$ assumed being applied to the pinion A while the viscous torque of resistance $M_B = (0, 0, -10 \varphi_B)^T$ is applied to the gear B . Gearwheels themselves assumed made of steel with Young's modulus $E_A = E_B = 2 \cdot 10^{11} \text{ Pa}$ and Poisson ratio $\nu_A = \nu_B = 0.3$, and have the same width, along the axis of rotation, of 0.1 m .

7.2 Dynamic transmission error

A value of the dynamic transmission error (DTE) has been chosen for the computational verification. If force of friction exists at contact then DTE is not constant. First of all let us introduce the auxiliary variable

$$\Delta = -r_{Ab} \varphi_A - r_{Bb} \varphi_B. \quad (19)$$

This value characterizes a mismatch for the base circles arc lengths. If teeth in pairs contacting have an ideal "rigid" unilateral constraints without compliance, and switching between teeth pairs is also ideal then the value of Δ has to be an identical zero.

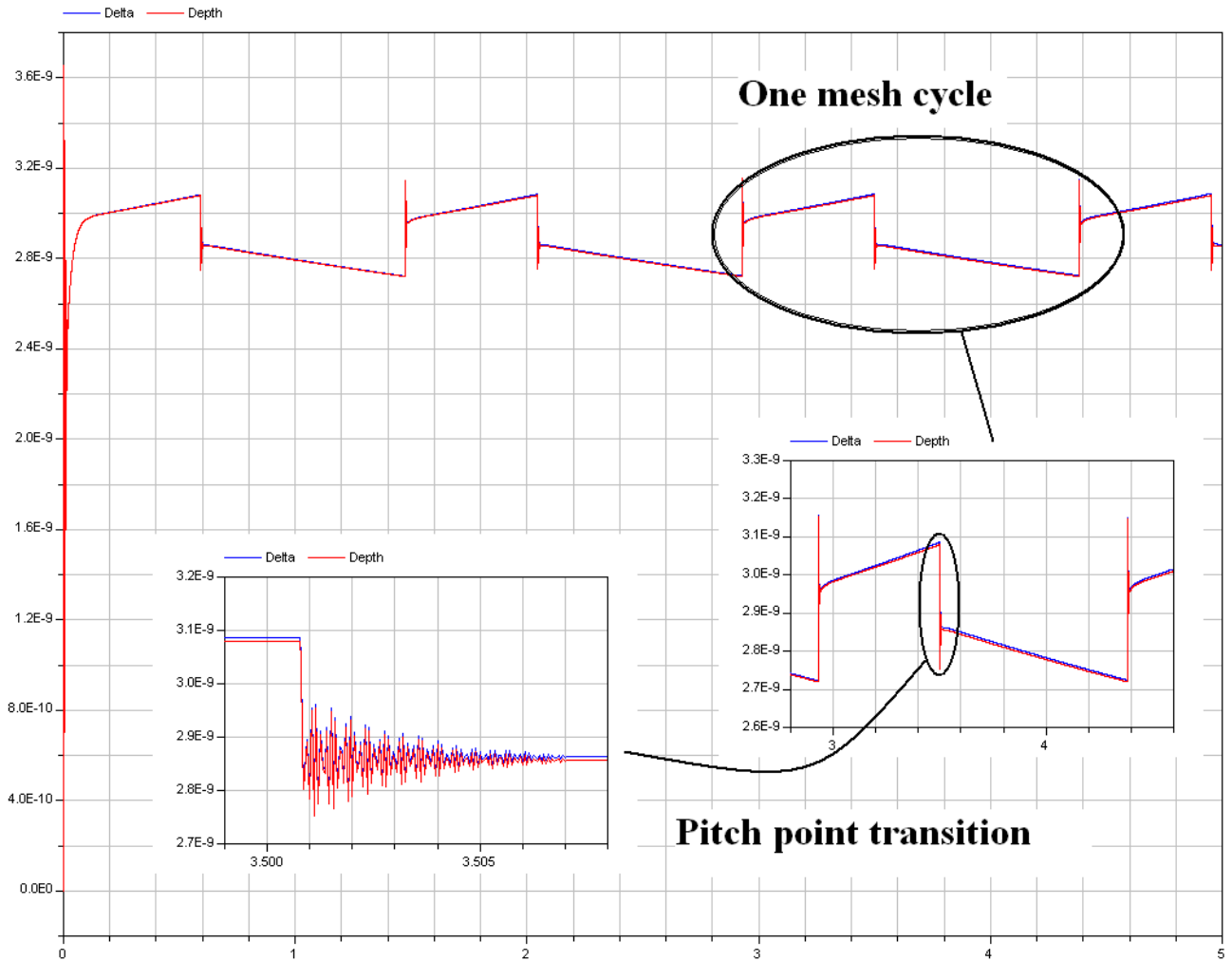


Figure 3: Comparison of the discrepancy (Delta) and the depth of penetration (Depth)

To undertake an analysis more detailed let us consider the results of numeric experiments. Firstly, one can remark that in case of the involute mesh a discrepancy $\Delta(t)$ from (19) has to be identical with the depth of mutual penetration of rigid teeth in vicinity of contact point. Indeed, the discrepancy (19) is exactly the difference between the arc distances of base circles while each of the wheels A and B rotates. This difference is accumulated from the very initial instant of simulation.

On the other hand it is known that the segment $P_A P_B$ is a perpendicular common to the teeth involutes penetrating each other, and simultaneously $P_A P_B$ lies exactly on the line $K_A K_B$ and its length is exactly the depth of teeth mutual penetration. Then there exists the only geometric possibility: the condition

$$|\overrightarrow{P_A P_B}| \equiv \kappa(t) \equiv \Delta(t)$$

has to be satisfied. The functions $\Delta(t)$, $\kappa(t)$ derived independently in the model are compared in Figure 3.

An effect obtained in an angular displacements due to pressing and subsequent penetrating in the Johnson contact model is certainly like one derived due to torsional deformations of elastic gearwheels [12, 13]. Moreover, finite element modeling do not disturb in any essential degree the whole dynamical picture if used to simulate teeth bending when contacting [14].

Indeed, one can compute the DTE according to the formula

$$\delta = -r_{Ab}\psi_A - r_{Bb}\psi_B \quad (20)$$

similar to (19). Here the values ψ_A , ψ_B are the angular displacements of the pinion and gear from their mean nominal positions $\Phi_A(t)$, $\Phi_B(t)$ such that the following equations fulfill

$$\varphi_\alpha(t) = \Phi_\alpha(t) + \psi_\alpha(t) \quad (\alpha = A, B).$$

These nominal values $\Phi_\alpha(t)$ correspond just to the case of rigid contact satisfying evidently the following

kinematic identity

$$-r_{Ab}\Phi_A(t) - r_{Bb}\Phi_B(t) \equiv 0.$$

Hence, it turns out in frame of our considerations that the identity

$$\Delta(t) \equiv \delta(t)$$

takes place.

Let us investigate now sources of the DTE presented in Figure 3. If one eliminates completely in the model the friction between teeth surfaces then the value $\delta(t)$ will grow asymptotically to its limit value thus providing a systematic error of transmission, see Figure 4, blue curve. The reason for this error is evidently a mutual penetration of the compliant contact model of Johnson.

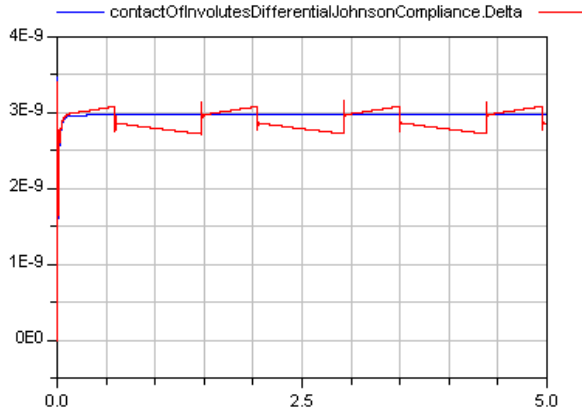


Figure 4: The DTE without (blue) and with (red) friction

If one introduces to the whole gearbox model, the simplest model of the Coulomb friction with the coefficient $f = 0.3$ then the systematic error will be superimposed by the periodic one, see Figure 4, red curve. This latter error has discontinuities at instants of the contact teeth pair changes and at the instant when the contact patch passes through the pitch point P , see Figure 1. The periodic DTE almost completely coincides with similar curves presented in papers [12, 13]. In the graphs of these papers one can note only small variations from exact curves of our Figures 3,4. An origin of these variations is evidently additional small deviation derived due to more exact account of the elastic torsion oscillations considered in [12, 13]. Additional splash of weak torsional oscillations one can observe in [12, 13] exists because of the multiplicity of teeth contacts in that model: time segments overlap for the mesh cycles of the nearby pairs. So when contact of the previous teeth pair vanishes then additional elastic disturbance arises. Remind that for simplicity we

Table 1: Comparative efficiency

Type of the contact model	Coefficient of friction	CPU time
Johnson's	0.3	20.4
Johnson's	0	16.1
rigid	0.3	13
rigid	0	11.4

consider in our model the case of mesh ratio which is equal to one.

7.3 Comparison with a rigid contact

A goal of our further numerical experiments is to compare two contact models when meshing: (a) the Johnson model, (b) the rigid contact model without compliance. The results of models under comparison simulation run showed for illustration in Figure 5. The normal contact force is counted along the y-axis of the plot. Case of the Johnson model corresponds to the blue curve while the red one is for the case of rigid contact model.

Comparison of the simulation results shows in Figure 5 that both contact models, the Johnson one and the rigid contact, bring the same dynamic result. The only difference is that the Johnson model generates additional oscillations of the normal contact force being superimposed on the normal force behavior for the rigid case. Moreover, one can see from detailing shown in Figure 5 that rigid contact model looks like a result of the procedure of averaging for the dynamics with the compliant contact, Johnson, model.

An advantage of the latter case is that this case of the contact model makes it possible to apply an arbitrary number of contacts for the body in the multibody system dynamics model without any restrictions. At the same time the rigid contact model does not allow such a possibility.

On the other hand, any contact model based on the FEM code application requires much more computational resources than in case of the "simple" compliant model analysed above. To compare an effectiveness of the Johnson contact model and the model using the rigid unilateral contact constraint consider Table 1 with preliminary relative estimations of the CPU time, in seconds, needed for both cases with addition of the friction force influence. Here the results of the simulation run are presented, without any optimization, for the model time of 5 seconds.

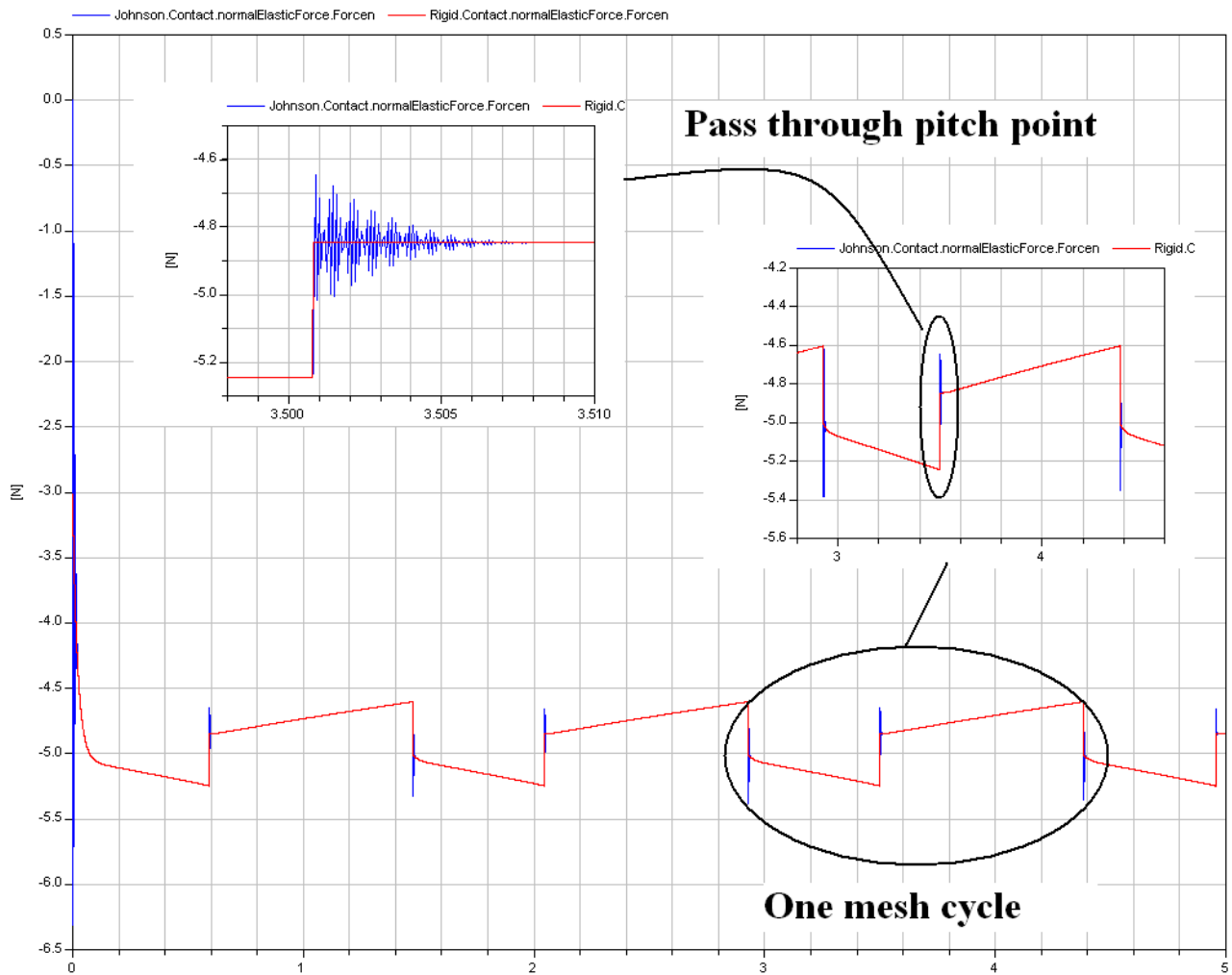


Figure 5: Comparison of the models with (blue) and without (red) compliance at contact

One can note easily from Table 1 that the Johnson model increases computational time in some degree, not larger than twice, in compare with the fastest case of the rigid contact. And simultaneously this model with compliance increases considerably the flexibility and universality of simulation tools in a wide range of applied problems.

8 Conclusions

Comparing our previous results with the above ones we can conclude that:

- since cylindrical contact models are restricted to the 2D-geometrical considerations they are simpler in a certain sense than the 3D-models;
- on the contrary, dynamical models became more complex in some degree because the Johnson model forces us to deal with the transcendental equation having a singularity at zero;

- involute meshing requires additional analytic efforts causing additional computational complexity increase;
- compliant models create an effect similar to one generated by the torsional elastic deformations of gearwheels;
- compliant model built up showed an efficiency high enough comparable with the fastest case of geometrically rigid constraint;
- the computer model built up makes it possible in an evident way to construct models of gearboxes of any complexity for the spur involute type of meshing.

9 Acknowledgements

The paper was prepared with partial support of Russian Foundation for Basic Research, projects 08-01-

00600-a, 08-01-00718-a, 08-08-00553-a.

References

- [1] Pelchen, C., Schweiger, C., Otter, M., Modeling and Simulating the Efficiency of Gearboxes and of Planetary Gearboxes. In: Proceedings of the 2nd International Modelica Conference, Deutsches Zentrum für Luft- und Raumfahrt e. V. (DLR), Oberpfaffenhofen, Germany, March 18–19, 2002, pp. 257–266.
- [2] Schlegel, C., Bross, M., Beater, P., HIL-Simulation of the Hydraulics and Mechanics of an Automatic Gearbox. In: Proceedings of the 2nd International Modelica Conference, Deutsches Zentrum für Luft- und Raumfahrt e. V. (DLR), Oberpfaffenhofen, Germany, March 18–19, 2002, pp. 67–75.
- [3] Schweiger, C., Otter, M., Modeling 3D Mechanical Effects of 1D Powertrains. In: Proceedings of the 3rd International Modelica Conference, Linköpings Universitet, Linköpings, Sweden, November 3–4, 2003, pp. 149–158.
- [4] 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.
- [5] 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.
- [6] Kossenko, I. I., Implementation of Unilateral Multibody Dynamics on Modelica. In: Proceedings of the 4th International Modelica Conference, Hamburg University of Technology, Hamburg–Harburg, Germany, March 7–8, 2005, pp. 13–23.
- [7] Kosenko, I. I., Alexandrov, E. B., Implementation of the Hertz Contact Model and Its Volumetric Modification on Modelica. In: Proceedings of the 6th International Modelica Conference, University of Applied Sciences Bielefeld, Bielefeld, Germany, March 3–4, 2008, pp. 203–212.
- [8] Litvin, F. L., Fuentes, A., Gear Geometry and Applied Theory. Cambridge University Press, Cambridge – New York – Melbourne – Madrid – Cape Town – Singapore – Sao Paulo, 2004.
- [9] Pereira, C. M., Ramalho, A. L., Ambrosio, J. A., A Critical Overview of Internal and External Cylinder Contact Force Models. Nonlinear Dynamics, Published Online: 18 September 2010.
- [10] Johnson, K. L., Contact Mechanics. Cambridge University Press, Cambridge, UK, 2001.
- [11] Förg, M., Engelhardt, T., Ulbrich, H. Comparison of Different Contact Models within Valve Train Simulations. Proceedings of ACMD2006, The Third Asian Conference on Multibody Dynamics 2006, Institute of Industrial Science, The University of Tokyo, Tokyo, Japan, August 1–4, 2006.
- [12] Vaishya, M., Singh, R., Sliding Friction–Induced Non–Linearity and Parametric Effects in Gear Dynamics. Journal of Sound and Vibration, 2001, Vol. 248, No. 4, pp. 671–694.
- [13] Vaishya, M., Singh, R., Strategies for Modeling Friction in Gear Dynamics. Journal of Mechanical Design, 2003, Vol. 125, Iss. 2, pp. 383–393.
- [14] He, S., Effect of Sliding Friction on Spur and Helical Gear Dynamics and Vibro–Acoustics. PhD thesis, The Ohio State University, 2008.

Import and Export of Functional Mock-up Units in JModelica.org

Christian Andersson^{a,c} Johan Åkesson^{b,c} Claus Führer^a Magnus Gäfvert^c

^aDepartment of Numerical Analysis, Lund University, Sweden

^bDepartment of Automatic Control, Lund University, Sweden

^cModelon AB, Sweden

Abstract

Different simulation and modeling tools often use their own definition of how a model is represented and how model data is stored. Complications arise when trying to model parts in one tool and importing the resulting model in another tool or when trying to verify a result by using a different simulation tool. The Functional Mock-up Interface (FMI) is a standard to provide a unified model execution interface. In this paper we present an implementation of the FMI specification in the JModelica.org platform, where support for import and export of FMI compliant models has been added. The JModelica.org FMI import interface is written in Python and offers a complete mapping of the FMI C API. JModelica.org also offers a set of Pythonic convenience methods for interacting with the model in an object-oriented manner. In addition, a connection to the simulation environment Assimulo which is part of JModelica.org is offered to allow for simulation of models following the FMI specification using state of the art numerical integrators. Generation of FMI compliant models from JModelica.org will also be discussed.

Keywords: JModelica.org; Assimulo; Sundials; FMI, FMUs

1 Introduction

In an effort to provide a unified model interface for different simulation tools and modeling environments, the MODELISAR consortium defined an open interface called the Functional Mock-up Interface. The idea is that both Modelica-based and non-Modelica-based tools may generate and exchange models that follow the FMI specification. FMI compliant models are referred to as Functional Mock-up Units (FMUs).

This enables users to create specialized models in one modeling environment, connect them in a second and finally simulate the complete system using a third simulation tool. This in turn, facilitates tool interoperability and model exchange.

In this paper, we present an implementation of the Functional Mock-up Interface in the JModelica.org platform, [2]. The implementation consists of support both for exporting FMUs from JModelica.org and importing FMUs generated by other tools.

Python was selected as the implementation language for the interface. The choice of Python for the integration was based on several reasons. The main advantage is that Python is a powerful and dynamic programming language with an clear and readable syntax with a low threshold for users to create their own simulation scripts, regardless of if you come from an MATLAB environment or a low-level programming language such as C. There are several packages that make Python a good option for scientific computing. One example is Scipy together with Numpy [13], which contains mathematically relevant functions, another is Matplotlib [10] for visualization in a MATLAB like format. There are also Python packages aimed specifically at interfacing code written in other programming languages. One such package is CTYPES [9], which makes connection to C and loading of dynamic linked libraries possible. This is a necessity for implementation of the FMI standard.

Another reason for choosing Python is that for part of the implementation, functionality could be reused with little or no modification as similar functionality already exists in JModelica.org. The XML framework needed to load and generate FMUs is already in-place in JModelica.org and the connection to the simulation environment Assimulo is similar to that of the connection between a model generated from JModelica.org

and Assimulo.

The paper is outlined as follows. In Section 2, a brief background is given about the JModelica.org platform and the simulation package Assimulo together with an overview of the Functional Mock-up Interface. The implementation is described in Section 3. In Section 4, the Van der Pol oscillator and the Full Robot from the Modelica Standard Library is simulated. The result is compared with a simulation using Dymola [6]. An example from the Air Conditioning library is also given. Finally, Section 6 summarizes this paper.

2 Background

2.1 JModelica.org

JModelica.org [2] is an "extensible Modelica-based open source platform for optimization, simulation and analysis of complex dynamic systems"¹ with the mission:

“To offer a community-based, free, open source, accessible, user and application oriented Modelica environment for optimization and simulation of complex dynamic systems, built on well-recognized technology and supporting major platforms.”

The platform offers compilers for Modelica, [15] and Optimica, [1], a simulation package called Assimulo and a direct collocation algorithm for solving large-scale DAE-based dynamic optimization problems. The user interface in JModelica.org is based on Python, which provides means to conveniently develop complex scripts and applications. The platform is designed both for large-scale industrial needs and for prototyping in a research environment. It provides synergies between state of the art methods resulting from research and problems industrially relevant problems. JModelica.org supports the major platforms Windows, Mac and Linux.

2.2 Assimulo

Assimulo, [3], is the default simulation package in JModelica.org. It is a Python package consisting of several solvers for solving explicit ordinary differential equations (ODEs),

$$\dot{x} = f(t, x), \quad x(t_0) = x_0 \quad (1)$$

¹<http://www.jmodelica.org>

as well as differential algebraic equations (DAEs),

$$\begin{aligned} F(\dot{x}, x, w, t) &= 0, \\ x(t_0) &= x_0, \quad \dot{x}(t_0) = \dot{x}_0, \quad w(t_0) = w_0. \end{aligned} \quad (2)$$

Examples of solvers supported by Assimulo are a fifth-order three-stage Radau method, explicit Euler with fixed step-sizes, and a fourth-order Runge-Kutta method. By interfacing to SUNDIALS [11], state-of-the art implementations of multistep methods for ODEs and DAEs are available through Assimulo. The solvers CVode and IDA in SUNDIALS are the latest development branch of codes implementing multistep methods dating back to the 80s, also including DASSL. CVode is a variable-order, variable-step multistep algorithm for solving ordinary differential equations. CVode includes the Backward Differentiation Formulas (BDFs), which are suitable for stiff problems as well as Adams-Moulton formulae for highly accurate simulation of non-stiff systems. The solver IDA is a DAE integrator based on BDF.

Assimulo consists of mainly two parts. First, a skeleton of a simulation problem, which allows for defining all the necessary methods needed for simulation of a hybrid ODE and DAE, such as the right-hand-side, root functions and time-events. These skeletons are defined in the `Explicit_Problem` and `Implicit_Problem` classes for the ODE and DAE case respectively. The second part contains the actual integrators and interprets the information from the problem specification and performs the simulation.

In order to use Assimulo together with JModelica.org, the problem classes from Assimulo needed to be extended to allow for handling of how the models are defined in JModelica.org. In Figure 1 an overview of the implementation is shown.

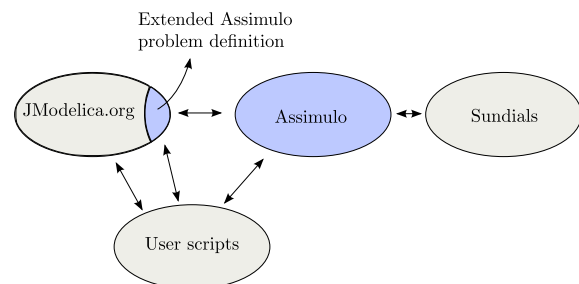


Figure 1: Overview of the interaction between JModelica.org, Assimulo and Sundials.

2.3 FMI

The Functional Mock-up Interface defines a standard for model exchange consisting of a set of C functions and an XML schema for model interaction. The mathematical formalism upon which FMI is based is that of hybrid ordinary differential equations (ODEs), i.e. ordinary differential equations with some discrete states. FMUs are distributed as compressed files containing:

- A shared object file (DLL), containing implementations of the FMI functions. In addition, or alternatively, the FMU may contain the source code corresponding to the compiled DLL.
- An XML file, containing the variable definitions and meta information for the model, together with information about how it was generated. The file also contains value-references for the variables, which uniquely identifies variables and which are used when retrieving data from the model.
- Optional files containing bitmaps, documentations, tables etc.

The C functions contained in the FMU are typically called by a simulation environment, in order to perform a simulation experiment. The simulation environment needs then to be able to handle simulation of hybrid ODEs, which are often stiff. Also, the model meta data contained in the XML file needs to be loaded by the simulation environment in order to extract model information, e.g. variable names.

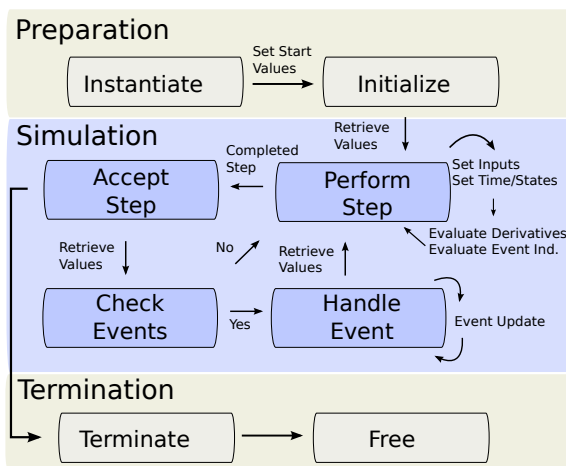


Figure 2: Overview of the calling sequence for an FMU.

In Figure 2, an overview of the calling sequence for an FMU is described. Prior to a simulation experiment, the model has to be instantiated. This includes

extracting the files in the FMU, loading the DLL and XML files and calling the instantiation function available in the DLL. A model can be instantiated multiple times for which the function `fmiInstantiateModel` is provided. The model is then initialized by calling `fmiInitialize`.

Once the model is instantiated and initialized it can be simulated. The simulation is performed by updating time and states in the model, via `fmiSetTime` and `fmiSetContinuousStates`, and by calculating the derivatives via `fmiGetDerivatives`.

During the simulation, events are monitored via the functions `fmiGetEventIndicators` and `fmiCompletedIntegratorStep`. State-events are detected by looking for sign changes in the event indicators while step-events are checked in the model after calling the completed step function when an integration step was successfully completed.

At an event, the function `fmiEventUpdate` has to be called. This function updates and re-initializes the model in order for the simulation to be continued. Information is also given about if the states have changed values, if new state variables have been selected and information about upcoming time events.

To retrieve or set variable data during a simulation, value-references are used as keys. All variables are connected to a unique number defined by the export tool and provided in the XML-file. This number can then be used to retrieve information about variables via functions in the interface or can be used to set input values during a simulation. There are functions for setting and getting values for Real, Integer, String and Boolean values, `fmi(Get/Set)(Type)`.

After a simulation, memory has to be deallocated. The function `fmiTerminate` deallocates all memory that have been allocated since the initialization and the function `fmiFreeModelInstance` dispose of the model instance.

3 FMI Implementation

3.1 Export

The FMI specification standardizes an execution interface for hybrid ODEs. Modelica models, on the other hand, are usually translated into systems of index-1 DAEs. Therefore, a Modelica-based tool needs to transform DAEs into ODE form. Starting with the DAE

$$F(\dot{x}, x, w, u, p, t) = 0 \tag{3}$$

where $x \in \mathbb{R}^{n_x}$ are the states, $w \in \mathbb{R}^{n_w}$ are the algebraic variables, $u \in \mathbb{R}^{n_u}$ are the inputs, $p \in \mathbb{R}^{n_p}$ are the parameters and $t \in \mathbb{R}$ is the time. The objective of this transformation is to obtain an ODE on the form

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

Notice that this transformation is conceptual in most cases in the sense that the function f cannot be computed explicitly. Rather, a commonly employed strategy is to regard the function F as a system of non-linear system equations, where \dot{x} and w are unknown and x , u , p and t are known. This strategy relies on the assumptions that i) start values for the states are available, possibly computed by solving a system of initialization equations, and ii) that the matrix

$$\begin{bmatrix} \frac{\partial F}{\partial \dot{x}} & \frac{\partial F}{\partial w} \end{bmatrix} \quad (5)$$

is square and has full rank. The latter condition means that a solution to the system of equations exists, at least locally, and holds if the DAE is of index 1, see [5] for a definition. Indeed, Modelica models commonly have index higher than 1, but it is here assumed that the index has been reduced by an index reduction algorithm, [12].

The DAE (3) is often highly structured and has in addition a sparse Jacobian, properties that can be exploited in order to perform the transformation more efficiently. A common approach for exploiting this structure is to decompose the DAE system into a sequence of smaller systems. This can be done by means of Tarjans algorithm, see [14, 8, 7] for details. Additional performance is gained typically as in typical cases several of the decomposed systems can be solved directly without the need to employ iterative Newton-type solvers. The usually few remaining non-linear systems of equations are solved during simulation by means of iterative techniques. In the FMUs generated by JModelica.org the KINSOL algorithm, which is part of the SUNDIALS suite, is used. In order to increase the robustness of the algorithm, KINSOL has been extended to support regularization to handle even the case of an initially singular Jacobian. See [16] for a detailed treatment. As a result of the presence of non-linear equation systems requiring iteration, ODE form (4) is conceptual rather implicit than explicit. Nevertheless, from the point of view of the simulation environment, the model is regarded as an explicit ODE, where the derivatives are computed given values of the parameters, the states, the inputs and the current time. The algorithm for computing the derivatives

as outlined above is made available in the FMI function `fmiGetDerivatives`. In addition to providing a function for evaluating the derivatives of the ODE, hybrid constructs resulting in events need to be supported in the simulation run-time system in the FMU. Examples of hybrid constructs in Modelica are instantaneous equations (expressed as `when`-clauses) and relational expressions. During continuous integration, a set of event indicator functions, provided in the function `fmiGetEventIndicators`, are monitored. If a sign change of one of the indicator functions is detected, an event has occurred and the simulation environment then informs the FMU by calling the function `fmiEventUpdate`. From the point of view of the simulation environment, this procedure is straight forward, since many integration algorithms provide native support for localization of events. For the internal simulation run-time system in the FMU, the situation is more complicated. For example, one event may trigger other events, which requires an event iteration scheme to be employed. In JModelica.org, the simulation run-time system performs a fixed point iteration to resolve dependent events. For more information about hybrid constructs in Modelica and how they are handled in the context of simulation, see [4].

Before an FMU can be simulated, consistent initial conditions need to be found. In the FMUs generated by JModelica.org, this is done similarly to how the derivatives are computed. The BLT transformation is applied to the initialization system, consisting of the index-1 DAE augmented by initial equations, in order to obtain a sequence of equation systems, which can be solved for the states, the derivatives and the algebraic variables. Also in this case, the modified version of KINSOL, supporting regularization, is used.

3.2 Import

Integration of the FMI to JModelica.org requires a few key features to be present.

- Ability to decompress a compressed archive.
- Ability to couple functions provided in a DLL to Python.
- Ability to read and interpret an XML-file.

These features are provided by use of several Python packages such as `ctypes` [9], `lxml` and `zipfile`. `ctypes` enables loading of a dynamic linked library (DLL) into Python. The functions of the DLL can then be retrieved and defined in Python which enables

them to be called directly. The functions have to be explicitly defined together with their arguments and return arguments so that the correct type is returned back to the DLL. `lxml` provides methods for handling of XML-files, such as querying and traverse complete files. This feature was already available in JModelica.org as an extended FMI XML format, which is used to handle generated model data from JModelica.org. Finally, `zipfile` offers methods for extracting information from compressed directories, such as an FMU.

The FMI import implementation in JModelica.org centers around a Python class, `FMUModel` where the constructor takes as input an FMU and performs the necessary tasks to enable manipulation and simulation of the FMU. The constructor also calls the XML import interface which reads the complete XML-file and populates data structures with information about all model variables including start-values, value-references, aliases and types. The interface consists of a raw mapping of the functions defined in the FMI specification easily available and accessible from Python. The methods are named according to the specification with a leading underscore. For example, the FMI function `fmiGetDerivatives(...)` corresponds to the following method in our Python class, `FMUModel`,

```
FMUModel._fmiGetDerivatives(...)
```

Providing a complete mapping to the original FMI functions enables users to create scripts tailored to their specific purposes.

JModelica.org also provides a Pythonic and object-oriented connection to an FMU with high-level methods for setting and retrieving values. We demonstrate this by computing the derivatives at time t and state y using the FMU:

```
FMUModel.time = t
FMUModel.continuous_states = y

rhs = FMUModel.get_derivatives()
```

The high-level methods propagate the information and call the underlying FMU functions.

To retrieve or set values of an arbitrary variable, instead of looking for the value reference, the name is used in the call to set/get methods:

```
FMUModel.get('der(x)')
FMUModel.set('g', 9.81)
```

The methods retrieve information about the variable, type and value reference from the XML data and then call the underlying FMU functions.

In addition to the high-level methods in JModelica.org, a connection to the simulation package Assimulo is also offered. The connection is based on extending Assimulo's problem class in the same way as models generated from JModelica.org are interfaced, see Figure 1.

As a problem class in Assimulo is just a skeleton of a model together with its methods, interfacing is just a matter of providing the information.

In order to connect the calculation of the derivatives of an FMU to Assimulo, the right-hand side function (*rhs*) must correspond to:

```
class FMIODE(Explicit_Problem):
    def f(t,y):
        #Moving data to the model
        FMUModel.time = t
        FMUModel.continuous_states = y

        #Evaluating the rhs
        rhs = FMUModel.get_derivatives()

    return rhs
```

where `Explicit_Problem` is Assimulo's skeleton class of an ODE problem. If there are any inputs, they are calculated and provided to the model as well.

Events are monitored and detected by providing methods for the state-, step- and time-events which in Assimulo correspond to implementing the methods `state_events`, `time_events` and `completed_step`. They are implemented similarly to the calculation of the derivatives.

If an event is detected, either be it state, time or step, a call is made to a method called `handle_event` in the problem which has been implemented so that it is directed to the FMI function `fmiEventUpdate`.

These methods provide a full-fledged connection to Assimulo and to state of the art numerical integrators. The current implementation is fully functional and relies on Sundials CVode solver.

4 Examples

4.1 The Van der Pol Oscillator

The Van der Pol oscillator stated in Equation (6) is used here to demonstrate the functionality of exporting an FMU using the JModelica.org platform and also to demonstrate the import process. In addition, it will be shown how the same problem is solved using Sundials CVode solver. The problem is also solved for a set of initial values to demonstrate how to run multiple simulations in a single sweep. Simulation of per-

turbed values can be useful for analyzing and evaluating model sensitivity with respect to uncertainty in physical parameters or initial conditions. The dynamics of the Van der Pol oscillator is given by

$$\frac{d^2x}{dt^2} - (1 - x^2)\frac{dx}{dt} + x = 0. \quad (6)$$

The problem can be described in the Modelica language by introducing the state variables $x_1 = x$ and $x_2 = \frac{dx}{dt}$, which gives

$$\begin{aligned} \frac{dx_1}{dt} &= x_2 \\ \frac{dx_2}{dt} &= (1 - x_1^2)x_2 - x_1. \end{aligned} \quad (7)$$

The Modelica specification for the Van der Pol oscillator is given in Listing 1, where the initial values are set to $x_1(t_0) = x_{1_0}$ and $x_2(t_0) = x_{2_0}$.

```
model VDP
  // The parameters
  parameter Real x1_0 = 1.0;
  parameter Real x2_0 = 0.0;

  // The states
  Real x1(start = x1_0);
  Real x2(start = x2_0);

  equation
    der(x1) = x2;
    der(x2) = (1 - x1^2) * x2 - x1;
end VDP;
```

Code Listing 1: The Van der Pol oscillator described in Modelica.

Creation of an FMU from a Modelica model consists of several steps. Primarily, the equations have to be translated and possibly manipulated by for instance an index reduction algorithm to produce a source code file, in our case a C file. Variable data needs to be extracted and populated into an XML structure. In JModelica.org, these steps are collected in a Python method, `compile_fm`, which is demonstrated below.

```
from jmodelica.fmi import compile_fm
fmu_name = compile_fm("VDP", "VDP.mo")
```

The commands produce an FMU of the Modelica model VDP located in `VDP.mo` which can be distributed to any software supporting the Functional Mock-up Interface.

Steps for simulating the Van der Pol oscillator are similarly straight forward where the FMU first must be loaded into JModelica.org.

```
from jmodelica.fmi import FMUModel
model = FMUModel(fmu_name)
```

`FMUModel` takes the name of an FMU, in our case `VDP.fmu` as an argument in the constructor. A number of internal steps are then taken when a model is loaded. First, the FMU is unzipped and the XML data together with the binary containing the model functions are extracted. Second, the functions in the model binary is connected to Python and instantiated according to the FMI specification.

Simulation of the Van der Pol oscillator is then performed by the `simulate` method.

```
result = model.simulate(final_time=10)
```

The Van der Pol oscillator is simulated from $t = 0.0$ to $t = 10.0$ using default options. In Listing 2, the runtime statistics is shown, which is printed in the prompt after a simulation, when solving the Van der Pol oscillator using JModelica.org and Assimulo together with the solver CVode (BDF).

```
Final Run Statistics: VDP
Nbr of Steps : 148
Nbr of Function Evaluations : 208
Nbr of Jacobian Evaluations : 3
Nbr of F-Eval During Jac-Eval : 6
Nbr of Root Evaluations : 0
Nbr of Error Test Failures : 11
Nbr of Nonlinear Iterations : 204
Nbr of Nonlinear Conv Failures : 0
```

Code Listing 2: Simulation statistics of the Van der Pol oscillator.

The result object returned from a simulation makes the simulation data and simulation trajectories easily available for either visualization or manipulation.

```
x1 = result["x1"]
x2 = result["x2"]
time = result["time"]
```

For visualization, the package Matplotlib [10] is used.

```
import matplotlib.pyplot as plt

plt.figure(1)
plt.plot(time, x1, time, x2)
plt.xlabel("Time [s]")
plt.legend(("x1", "x2"))
plt.title("Van der Pol")
plt.show()
```


The resulting trajectories for x and $\frac{dx}{dt}$ are shown in Figure 3.

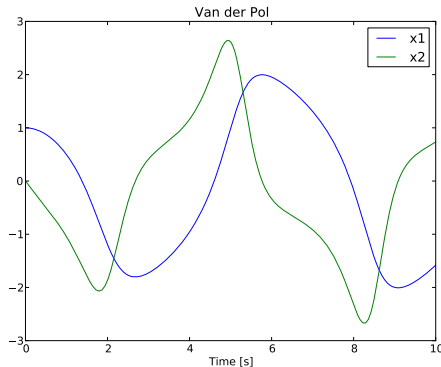


Figure 3: Solution of the Van der Pol oscillator showing x and $\frac{dx}{dt}$.

An extension of performing a simulation of the Van der Pol oscillator is to perform a parameter sweep. A parameter sweep is executed by varying a parameter and performing a simulation for each value. In our case, we consider holding $x_1(t_0)$ fixed while varying $x_2(t_0)$ in the interval $[-3, 3]$. The resulting script is shown below.

```
import numpy

nbr_points = 11
x1_0 = 0.0
x2_0 = numpy.linspace(-3.0, 3.0, 11)

for i in range(nbr_points):

    model.set('x1_0', x1_0)
    model.set('x2_0', x2_0[i])

    result = model.simulate(final_time=20)

    x1=result['x1']
    x2=result['x2']

    plt.plot(x2, x1, 'b')

plt.title("Van der Pol")
plt.ylabel("x1")
plt.xlabel("x2")
plt.grid()
plt.show()
```

First, the initial values are defined as $x_{1_0} = 0.0$ and x_{2_0} being a uniformly distributed array in the interval $[-3, 3]$ with 11 values. Second, the simulation command is iterated over the initial values which are set with the `model.set()` method. In each iteration, the model is simulated from 0.0 to 20 seconds and the solution trajectories for x_1 and x_2 are retrieved and plot-

ted. The resulting phase plot is shown in Figure 4.

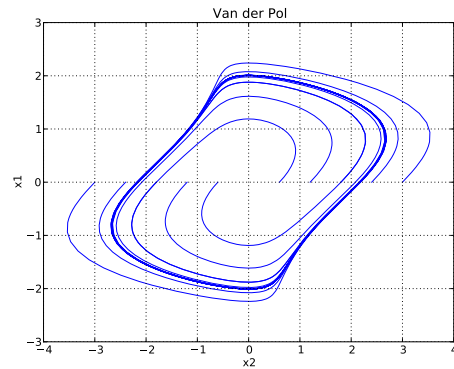


Figure 4: Solution of the Van der Pol oscillator showing a phase plot of x and $\frac{dx}{dt}$.

4.2 A Robot Model

The Full Robot from the multibody examples in the Modelica Standard Library will be used to demonstrate that the implementation can handle industrially relevant problems. The example is also intended to demonstrate that JModelica.org is able to simulate models generated by third party software supporting the FMI specification. In Figure 5, the diagram layer of the robot is depicted.

The robot consists of brakes, motors, gears and path planning. The model consists of 36 continuous states and around 700 algebraic variables together with 98 event indicators and a few thousand constants/parameters.

The FMU was generated using Dymola 7.4 [6].

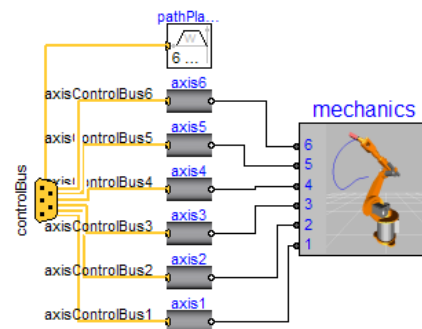


Figure 5: Overview of the Full Robot.

Simulation of the robot is performed following the same steps as in the Van der Pol example by first loading the model² and then invoking the simulate method.

²The name of the FMU has been shortened to save space in the article.


```

from jmodelica.fmi import FMUModel

robot=FMUModel('Modelica...fullRobot.fmu')
result = robot.simulate(final_time=1.8)
    
```

The robot is simulated from $t = 0.0$ to $t = 1.8$ using default options. In Listing 3, the run-time statistics is shown.

```

Final Run Statistics: Modelica_..._fullRobot

Nbr of Steps                : 1834
Nbr of Function Evaluations : 2386
Nbr of Jacobian Evaluations : 65
Nbr of F-Eval During Jac-Eval : 2340
Nbr of Root Evaluations    : 2223
Nbr of Error Test Failures  : 42
Nbr of Nonlinear Iterations : 2202
Nbr of Nonlinear Conv Failures : 0
    
```

Code Listing 3: Simulation statistics of the Full Robot using JModelica.org.

Trajectories for the joint velocities are extracted from the result object and visualized using Matplotlib in the same way as in the Van der Pol example.

```

dq1 = result['der(mechanics.q[1])']
dq6 = result['der(mechanics.q[6])']
time = result['time']

import matplotlib.pyplot as plt

plt.plot(time, dq1, time, dq6)
plt.legend(['der(mechanics.q[1])',
           'der(mechanics.q[6])'])
plt.xlabel('Time (s)')
plt.ylabel('Joint Velocity (rad/s)')
plt.title('Full Robot')
plt.show()
    
```

The result of the simulation is shown in Figure 6.

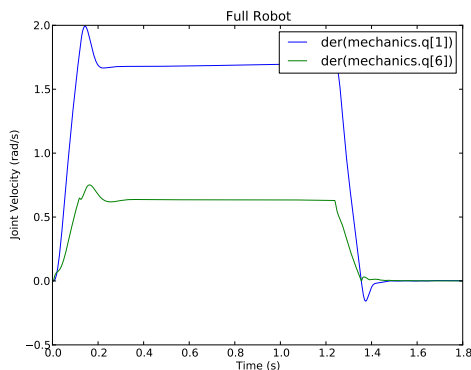


Figure 6: Solution of the joint velocities $q[1]$ and $q[6]$.

4.3 Verification of the Full Robot

For verification of the result, the Robot is simulated using Dymola 7.4 and the trajectories are compared. In Listing 4, the run-time statistics is shown when simulating the robot using Dymola and the solver DASSL.

```

Number of result points      : 1001
Number of GRID points       : 1001
Number of (successful) steps : 1482
Number of F-evaluations     : 10562
Number of H-evaluations     : 2794
Number of Jacobian-evaluations : 353
    
```

Code Listing 4: Simulation statistics of the Full Robot using Dymola.

In Figure 7 the resulting comparison between the simulation result from JModelica.org and Dymola is shown. The simulations are both performed with a relative tolerance of 10^{-4} and the absolute tolerance (in JModelica.org) was set to 0.01 times the relative tolerance times the nominal values of the continuous states, $0.01 \cdot \text{rtol} \cdot \text{nominal}$. The number of output points is set to 1000 in both cases.

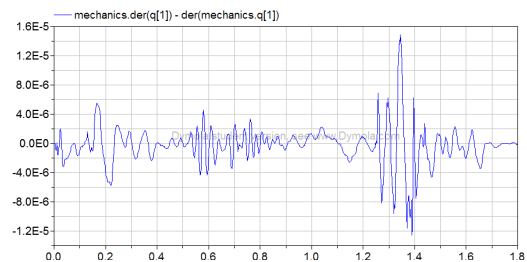


Figure 7: Difference of the state $\text{mechanics.der}(q[1])$ (joint velocity) between JModelica.org and Dymola. The model is simulated using default tolerances together with 1000 output points.

Timing results are shown in Table 1. Dymola native corresponds to simulating the robot directly from the Modelica standard library without using the FMI. Dymola FMU corresponds to loading a generated FMU into Dymola and performing a simulation. JModelica.org corresponds to the actual integration time and the time for I/O operations for storing the result.

Platform	Simulation Time
Dymola Native	1.26 s
Dymola FMU	4.97 s
JModelica.org	3.49 s

Table 1: Benchmark results of the Full Robot.

4.4 Twin Evaporator

The TwinEvaporatorCycle model from the commercial Air Conditioning library is demonstrated to show that the implementation can handle large models. It is a model of an A/C-cycle from a typical European premium car with individual front and rear climate zones. The model describes the cooling performance of the refrigerant cycle, and includes the compressor, front condenser, expansion control valve and two evaporators. The front evaporator is located under the dashboard and cools air for the driver and front seat passenger, while the rear evaporator is located between the seats and cools the air that flows to the back seat. The model diagram, shown in Figure 8, also includes dynamical display components for simulation analysis, e.g. the pressure-enthalpy diagram of the refrigerant R134a.

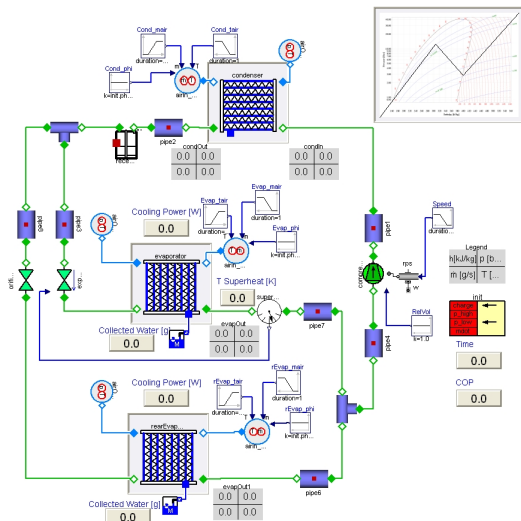


Figure 8: Overview of the Twin Evaporator.

The model consists of 130 states together with 1090 event indicator functions. The number of parameters and constants is close to 20.000, resulting in a model description file of 170.000 lines.

The model was simulated using JModelica.org from $t = 0.0s$ to $t = 180.0s$ with a relative tolerance of $rtol = 10^{-5}$ and an absolute tolerance corresponding to $atol = 0.01 \cdot rtol \cdot nominal$, as in the Full Robot example. The simulation statistics from JModelica.org can be found in Listing 5.

```

Final Run Statistics: ..._TwinEvaporatorCycle
Nbr of Steps                : 827
Nbr of Function Evaluations : 1434
Nbr of Jacobian Evaluations : 39
Nbr of F-Eval During Jac-Eval : 5070
Nbr of Root Evaluations    : 830
Nbr of Error Test Failures  : 44
    
```

```

Nbr of Nonlinear Iterations : 1422
Nbr of Nonlinear Conv Failures : 8
    
```

Code Listing 5: Simulation statistics of the TwinEvaporatorCycle using JModelica.org.

The model was also simulated with Dymola using the same options of the tolerances. The result can be found in Listing 6. Note that the Dymola simulation was performed on the Modelica model, not the FMU.

```

Number of result points : 1004
Number of GRID points : 1001
Number of (successful) steps : 302
Number of F-evaluations : 3859
Number of H-evaluations : 1303
Number of Jacobian-evaluations : 136
    
```

Code Listing 6: Simulation statistics of the TwinEvaporatorCycle using Dymola.

In Table 2, timing results of simulations with both JModelica.org and Dymola are listed. The simulation time corresponds to the actual integration time, including writing the result. The total time also includes time for the initialization.

Platform	Simulation Time	Total Time
Dymola Native	57.6 s	79 s
Dymola FMU	125 s	175 s
JModelica.org	90 s	130 s

Table 2: Benchmark results of the Twin Evaporator.

The execution time measurements indicate that the performance of the JModelica.org FMU import is on par with state of the art commercial tools.

5 Limitations

While fully FMI compliant FMUs are generated by JModelica.org, both in terms of the DLL functions and in terms of the XML files, the Modelica language compliance of the compiler front-end is not complete. The support is continuously improving and recent additions include support for hybrid and sampled systems.

The FMI standard specifies how several FMUs can be simulated jointly by connecting their inputs and outputs. One application of this feature is to include FMUs in Modelica models, and another application is co-simulation. These features remains to be implemented.

6 Summary

In this paper, an implementation of the Functional Mock-up Interface for Model Exchange in the JModelica.org platform has been presented. The export functionality enables users to generate FMI compliant models, FMUs from Modelica models and to use them in different FMI compliant tools.

The FMU import is based on Python. Models can be imported into the Python environment, where FMUs are represented by objects of a Python class. The FMU model class provides the user with a one to one mapping of the FMI functions, as well as convenient high-level methods for setting parameter values and simulating models.

This paper also shows that simulation of Functional Mock-up Units using JModelica.org produce results comparable to those produced by a state of the art commercial tool.

Future extensions include support for sparse Jacobians in the FMI specification.

This work was partially funded by the ITEA2 project OPENPROD.

References

- [1] Johan Åkesson. Optimica—an extension of modelica supporting dynamic optimization. In *In 6th International Modelica Conference 2008*. Modelica Association, March 2008.
- [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 problem. *Computers and Chemical Engineering*, 34(11):1737–1749, November 2010. Doi:10.1016/j.compchemeng.2009.11.011.
- [3] Christian 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.
- [4] Willi Braun, Bernhard Bachmann, and Sabina Pross. Synchronous events in the OpenModelica compiler with a petri net library application. In *3rd International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools*, 2010.
- [5] K.E. Brenan, S.L. Campbell, and L.R. Petzold. *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations*. Society for Industrial and Applied Mathematics, 1996.
- [6] Dassault Systemes. Dymola web page, 2010. <http://www.3ds.com/products/catia/portfolio/dymola>.
- [7] Hilding Elmqvist. *A Structured Model Language for Large Continuous Systems*. PhD thesis, Department of Automatic Control, Lund University, Sweden, May 1978.
- [8] Jan Eriksson. A note on the decomposition of systems of sparse non-linear equations. *Bit Numerical Mathematics*, 16(4):462–465, 1976. DOI: 10.1007/BF01932730.
- [9] Python Software Foundation. ctypes: A foreign function library for Python, 2009. <http://docs.python.org/library/ctypes.html>.
- [10] J. Hunter, D. Dale, and M. Droettboom. Matplotlib: Python plotting, 2010. <http://matplotlib.sourceforge.net/>.
- [11] Center for Applied Scientific Computing Lawrence Livermore National Laboratory. SUNDIALS (SUite of Nonlinear and Differential/ALgebraic equation Solvers), 2009. <https://computation.llnl.gov/casc/sundials/main.html>.
- [12] Sven Erik Mattsson and Gustaf Söderlind. Index reduction in differential-algebraic equations using dummy derivatives. *SIAM J. Sci. Comput*, 14(3):677–692, May 1993.
- [13] T. Oliphant. Numpy Home Page, 2009. <http://numpy.scipy.org/>.
- [14] R.E Tarjan. Depth-first search and linear graph algorithms. *SIAM J. Computing*, 1(2):146–160, 1972.
- [15] The Modelica Association. The Modelica Association Home Page, 2010. <http://www.modelica.org>.
- [16] Johan Ylikiiskilä, Johan Åkesson, and Claus Führer. Improving Newton’s method for initialization of Modelica models. In *8th International Modelica Conference*, 2011.

Implementation of Modelisar Functional Mock-up Interfaces in SimulationX

Christian Noll, Torsten Blochwitz, Thomas Neidhold, Christian Kehrer
 ITI GmbH
 Webergasse 1, 01067 Dresden, Germany
noll@iti.de, blochwitz@iti.de, neidhold@iti.de, kehrer@iti.de

Abstract

This document describes the implementation of the Modelisar Functional Mock-up Interfaces (FMI) in SimulationX. It presents the code generation of Functional Mock-up Units (FMU) for Model Exchange and Co-Simulation as well as the import of an FMU into SimulationX.

Keywords: Simulation; Modelisar; Functional Mock-up Unit (FMU); Functional Mock-up Interface (FMI)

1 Introduction

FMI stands for “*Functional Mock-up Interface*” [1] and was specified in the ITEA2 **Modelisar** project [2]. The intention is that dynamic system models of different software systems can be used together for software/model/hardware-in-the-loop simulation and for embedded systems. Using SimulationX Code Export, the functionality of a complete simulation model can be transformed into an FMU (*Functional Mock-up Unit*), which implements the FMI (*Functional Mock-up Interface*). A so created FMU can be instantiated by SimulationX or another simulation tool and accessed via the FMI functions. An FMU may either be self-integrating (*Co-Simulation*) or require the simulator to perform the numerical integration (*Model Exchange*).

2 FMU Support in SimulationX

There are two different FMI specifications (see Figure 1: FMI specifications), *FMI for Model Exchange* and *FMI for Co-Simulation*. Both are supported by SimulationX.

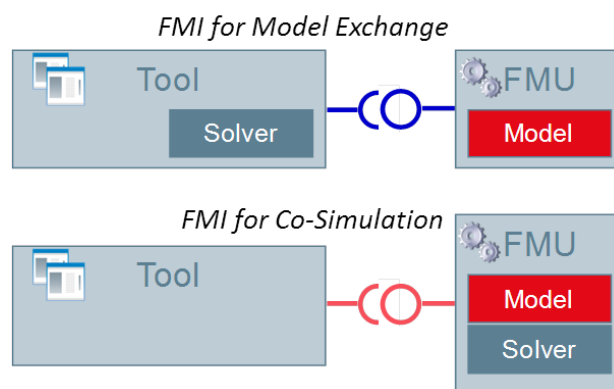


Figure 1: FMI specifications

2.1 FMU Code Export

Using SimulationX Code Export, the functionality of a complete simulation model can be transferred into an FMU (*Functional Mock-up Unit*). An FMU is distributed in the form of a zip File (*.fmu) and consists basically of the following components:

1. Exported Model + Interface

The exported model functionality is accessible through standardized C-functions (FMI). By using the programming language C high portability is guaranteed. This component can be present as pure source code or as a binary (DLL). The FMI-Interface includes:

 - Functions for instantiation, initialization, termination and destruction.
 - Support of Real, Integer, Boolean and String inputs, outputs and parameters.
 - *Set* and *Get* functions for each type, e.g. *fmiSetReal(...)*.
 - Functions for exchange of simulation data, e.g. *fmiGetDerivatives(...)*

There is no explicit function call for the computation of the model algorithm. The FMU decides on its own, depending on which data have been set and the data being sought, which calculation is initiated. For efficiency it is important that variables are not newly computed, if they have been computed already at an earlier step. Instead they shall be reused. This feature is called “caching of variables” in the sequel.

representing a model which may then be easily integrated into another simulation environment.

The following illustration (see Figure 2: FMI Code Export for Model Exchange) shows the schematic workflow for transferring a SimulationX model into an *FMU for Model Exchange*.

After all desired inputs, outputs and parameters have been defined by the user in the Code Export Wizard, the code export process starts.

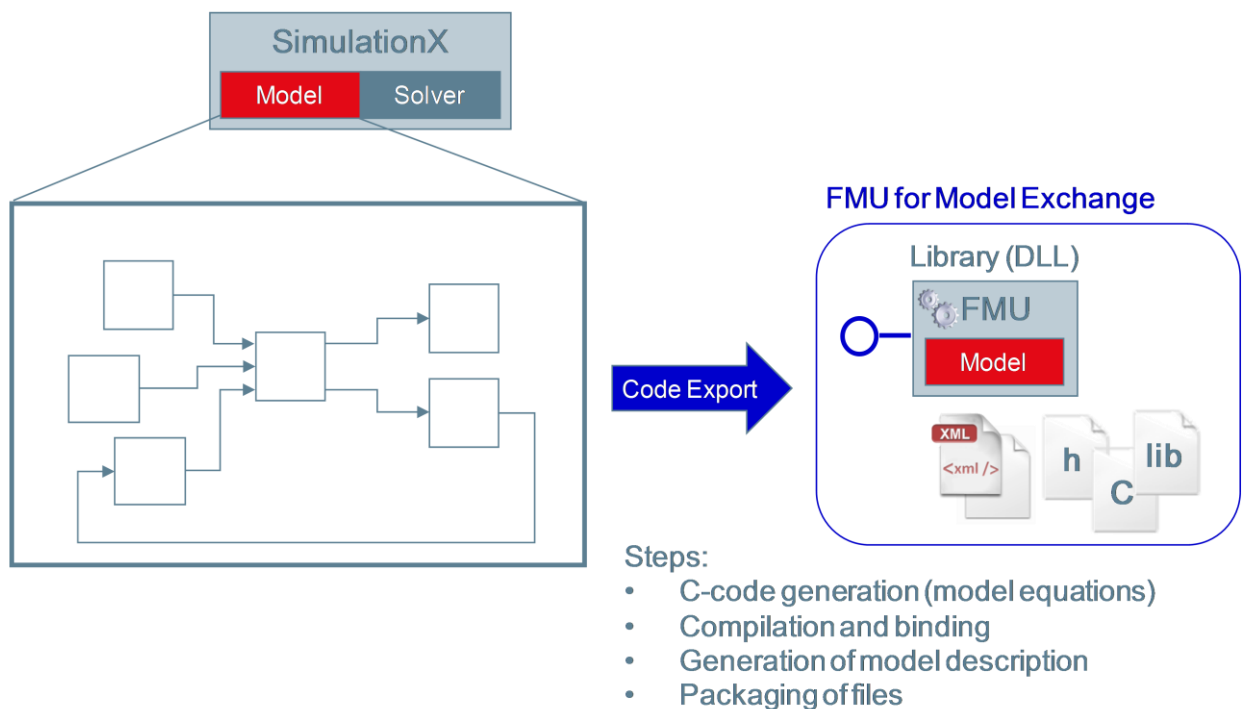


Figure 2: FMI Code Export for Model Exchange

2. Model Description Scheme

This scheme is represented by an XML file that contains the description of the required data for the information flow between the FMU and the simulation tool. Through the description of the model within an XML file, the provider of simulation tools are not forced to use a specific representation of data structures.

3. Data and Documentation (optional)

Additional data and documentation of the model can be included.

During the code export the following steps are executed. At first a special symbolic analysis will transfer the model into ordinary differential equations. Based on this equations and the defined interface, the C-code that includes the model functionality and the specific FMI interface functions, is generated. Furthermore the XML model description file is generated. At the end of this process a zip-file (*.fmu), with all necessary files, is created to distribute the FMU.

2.1.1 FMI for Model Exchange

The intention of *FMI for Model Exchange* is to allow any modeling tool to generate C code or binaries

2.1.2 FMI for Co-Simulation

The FMI for Co-Simulation is an interface standard for the solution of time dependent coupled systems consisting of subsystems that are continuous in time

(model components that are described by differential equations) or time-discrete (model components that are described by difference equations like, e.g. discrete controllers).

The FMI for Co-Simulation defines interface routines for the communication between a master and the individual simulation tools (*slaves*) in the co-simulation environment. The data exchange is restricted to discrete communication points in time and the subsystems are solved independently between these communication points.

The following illustration (see Figure 3: FMI Code Export for Co-Simulation) shows the schematic workflow to transfer a SimulationX model into an *FMU for Co-Simulation*.

simulation environment, is provided. In particular, this includes a set of capability flags to characterize the ability of the slave to support advanced master algorithms. One of these flags is *canHandleVariableCommunicationStepSize* that specifies whether the slave can handle variable communication step size. Another flag is *canRejectSteps* that indicates the slave's capability to discard and repeat a communication step. This will be supported in a future SimulationX release.

The flag *canInterpolateInputs* defines that the slave is able to interpolate continuous inputs. In this case, calling of *fmiSetRealInputDerivatives(...)* has an effect for the slave. At the end of the export process a zip-file (*.fmu) is created to distribute the FMU.

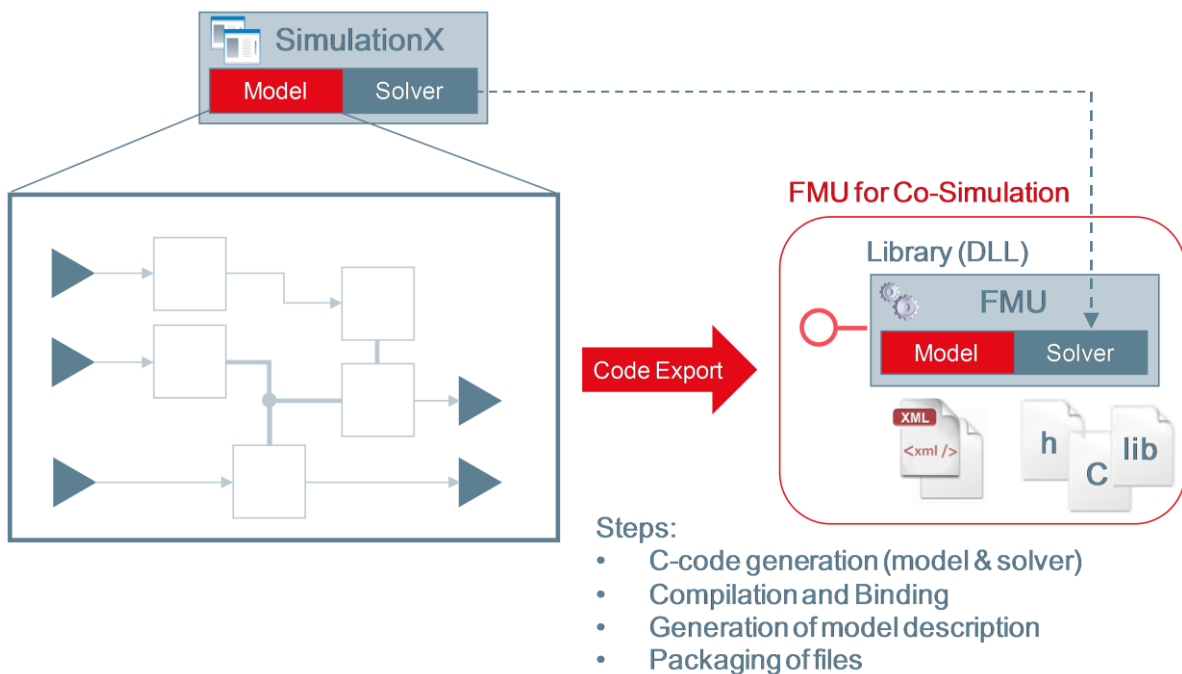


Figure 3: FMI Code Export for Co-Simulation

After all desired inputs, outputs and parameters have been defined by the user in the Code Export Wizard the code export process starts. During the Code Export the following steps are executed: At first a special symbolic analysis will transfer the model into ordinary differential equations. Based on this equations and the defined interface, the C code, that includes the model functionality, the specific FMI interface functions and a Solver (CVODE), is generated. The Sundials CVODE solver [4] uses a BDF variant and is well suited for stiff models.

Furthermore the XML model description file is generated, where all information about the slaves, which is relevant for the communication in the co-

2.2 FMU Import

The SimulationX FMU import consists of unzipping the *.fmu file and the generation of Modelica code including the calls of FMI functions based on the XML model description. A re-export via code export is supported.

The main idea of embedding a FMU into a Modelica model is to construct an external object and some external functions to interact with that model.

The automatic import process is started by selecting the menu *Insert/Functional Mockup Unit....*

Thereupon the following dialog (see Figure 4: FMU Import Dialog) for importing a FMU appears.

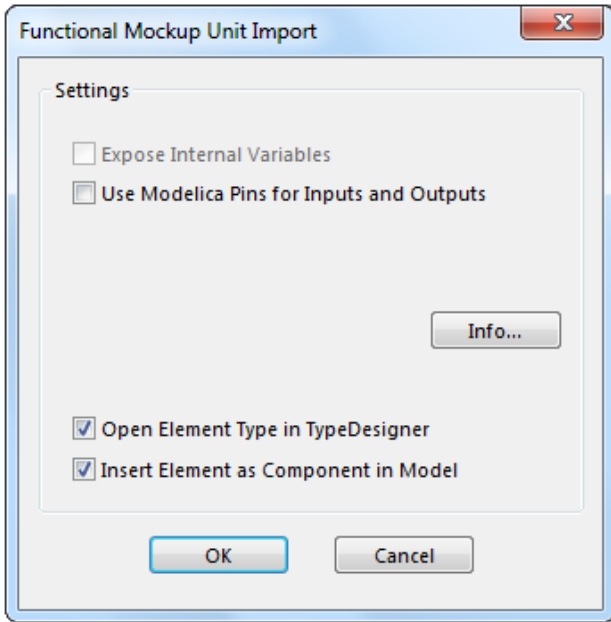


Figure 4: FMU Import Dialog

During the import process the DLL and Lib files (if any exist) are copied to the External Function folder.

2.3 Tool Coupling

The current version 1.0 of the FMI for Co-Simulation standard not only allows the coupling of specially prepared software modules (FMU), but can also be used for direct coupling of CAE tools. Thereto the particular application with its proprietary interfaces is made available via a special wrapper (see Figure 5: Tool coupling via wrapper DLL) that implements the standardized Functional Mockup Interface and provides it for other applications. From the outside, the particular application behaves like a Functional Mock-up Unit.

For SimulationX such a wrapper will be available. The implementation is based on the existing COM interface of SimulationX. For integrating a Modelica model in such a co-simulation an adequate preparation is necessary. Especially the inputs, outputs and parameters of the model have to be defined. All this information is stored as a "real FMU" in a zip archive. The model itself or a link to this model in the local file system or on the network must also be stored in this file.

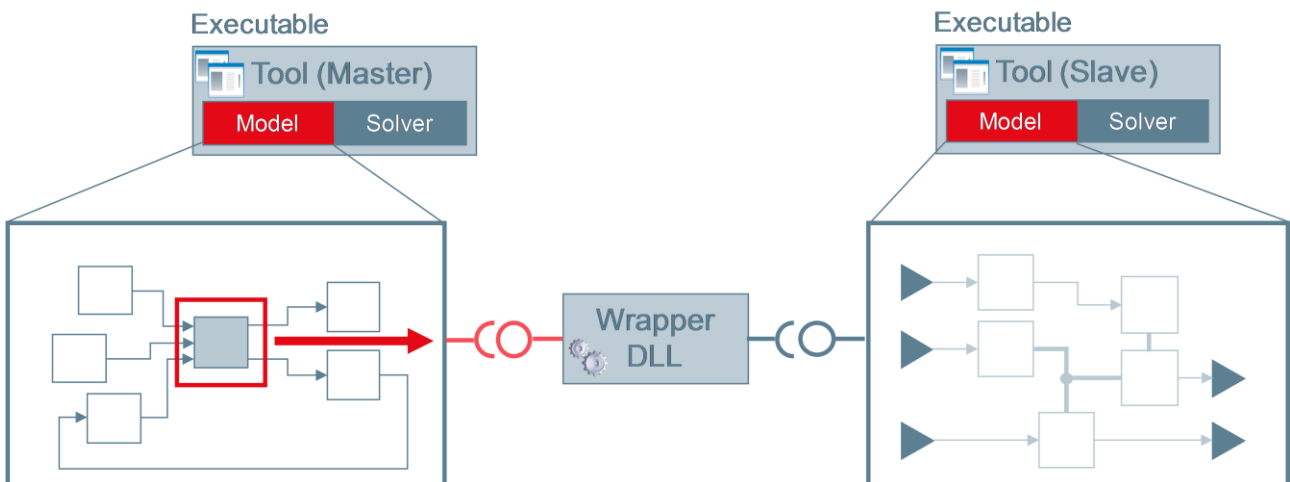


Figure 5: Tool coupling via wrapper DLL

To link an FMU with a Modelica model SimulationX uses an *External Object*. The *fmiInstantiateModel(...)/ fmiInstantiateSlave(...)* and *fmiFreeModelInstance(...)/ fmiFreeSlaveInstance(...)* functions are called as constructor and destructor, respectively. All other functions are called as external functions with an external object as first parameter. Because the *fmiInitialize(...)* function has to forward function pointers for several purposes, this function is redirected through a special built-in function.

3 Implementation Difficulties

During the implementation of the FMU import into SimulationX as a Modelica simulator a few difficulties had to be overcome.

The first problem is related to a type mismatch between *fmiBoolean* and Modelica *Boolean*, which leads to a type cast for scalar *fmiBoolean* variables and the necessity of restoring *fmiBoolean* arrays.

Further, there is an issue concerning the *fmiInstantiateModel* function, because the argument *fmiCallbackFunctions* is a *struct* that holds function pointers. In Modelica there is no possibility to generate such a record.

Also, it is not easy to implement that the function *fmiInitialize* is called only once, because according to Modelica language specification the body of a *when initial()* clause may be traversed several times during initialization.

Changing of discrete variables is only allowed in Modelica at event steps, not during continuous integration, but the *fmiSetXxx* functions returns *fmiStatus* as a Modelica integer variable, which is a discrete variable. Hence a Modelica compiler may call such functions only at event time instances. But the *fmiSetXxx* functions have to be called during continuous integration too.

The functions *fmiCompletedIntegratorStep*, *fmiEventUpdate*, and *fmiTerminate* are impure and thus may not be treated like constant functions. But, there is no possibility in Modelica to mark a function as impure.

There are two difficulties related to the FMI calling sequence. First, there is no possibility in Modelica to be informed about the reason for a model computation. But, it is relevant to distinguish between calling for instance *fmiEventUpdate* or *fmiCompletedIntegratorStep*. Secondly, Modelica does not provide the functionality to trigger an event step and call *fmiEventUpdate*.

Modelica has no functionality to provide event indicators (*evi*) directly. According to FMI specification FMUs have to add a small hysteresis to the *evi*. A Modelica tool may do the same with its internal root functions. Hence the hysteresis is added twice and events caused by the FMU are located a little bit inaccurately.

We solved these problems by using some internal Modelica extensions in SimulationX, which we also proposed to the Modelica Language Design Group and accordingly to the FMI standard committee.

4 Conclusions

With the Modelisar FMI standard exists a vendor-neutral interface that allows the exchange of simulation models between different tools and platforms. The chances to establish FMI as a standard are pretty good, because software vendors and users were involved right from the start. At the end, the success of

this interface is measured by how the tool vendors will integrate FMI into their products. In addition to reliability and numerical stability the ease of use will determine this success.

References

- [1] Functional Mock-up Interface: <http://www.functional-mockup-interface.org/index.html>
- [2] ITEA2 Modelisar Project: <http://www.modelisar.com>
- [3] Arnold M., Blochwitz T., Clauß C., Neidhold T., Schierz T., Wolf S., FMI for Co-Simulation: Multiphysics Simulation - Advanced Methods for Industrial Engineering. Bonn, June 2010.
- [4] M. Otter, T. Blochwitz, H. Elmqvist, A. Junghanns, J. Mauss, H. Olsson: Das Functional Mockup Interface zum Austausch Dynamischer Modelle. Plenary talk at the ASIM workshop. Ulm, 4. - 5. March 2010.
- [5] Neidhold T. Tool Independent Model Exchange Based on Modelisar FMI. Industrietag Informationstechnologie. Halle(Saale), May 2010.
- [6] SUNDIALS: <https://computation.llnl.gov/casc/sundials/main.html>.
- [7] FMI for Model Exchange v1.0: http://www.functional-mockup-interface.org/specifications/FMI_for_ModelExchange_v1.0.pdf
- [8] FMI for Co-Simulation v1.0: http://www.functional-mockup-interface.org/specifications/FMI_for_CoSimulation_v1.0.pdf

Experiences with the new FMI Standard Selected Applications at Dresden University

Christian Schubert^a Thomas Neidhold^b Günter Kunze^a

^aDresden University of Technology, Chair of Construction Machines and Conveying Technology
Münchner Platz 3, 01187 Dresden, Germany

^bITI GmbH, Webergasse 1, 01067 Dresden, Germany

January 31, 2011

Abstract

This paper describes how the new FMI standard is used at Dresden University to provide a universal model exchange. By employing the new standard exchange format a single simulation model can be used for different purposes in different software tools. Specific applications and their advantages will be presented as well as the development effort for their realisation. Special attention is paid to how FMI enables SimulationX models to be run in an interactive simulation on a motion platform. Finally, the benefits and drawbacks of the FMI interface are discussed from a user's perspective.

Keywords: FMI, Modelisar, Virtual Reality, SimulationX, Motion Platform, SARTURIS

1 Introduction

Simulation has become an indispensable tool in the field of machine construction and design. It is especially beneficial for Heavy Mobile Machinery since their low quantities and high production costs render real prototypes uneconomical. During the design process, there are many tasks which can be supported by simula-

tion such as comparing the effect of different parameter combinations and model variations.

In order to achieve maximum efficiency, verified models should be reused as often as possible. However, this can only be possible if there exists a common standardised interface for model exchange between different software tools for different applications.

Modelica has already proven to be a valuable contribution towards the simulation of Mobile Machinery [1, 2]. On the one hand it features an inherent support of multidomain models, on the other hand it features a seamless transition between a mathematical and a graphical model definition allowing for complex but structured models at the same time. Although Modelica is tool independent and can thus be used to exchange models between certain simulation environments, it cannot be used as a general model interface. The reason is, that every tool used for import would have to parse, flatten and optimise the Modelica code for which no general purpose library is available. The new Functional Mockup Interface (FMI), a result of the MODELISAR project [3], overcomes the aforementioned problem.

The next chapter features a brief description of the FMI standard. It is followed by an

overview of the work and the employed software tools at the Chair of Construction Machines and Conveying Technology at Dresden University before FMI has been introduced. Chapter four presents the benefits which have been achieved by implementing and applying FMI to this software environment. Special attention is paid to what has been changed and what had to be implemented. One major achievement of integrating the FMI standard is the option to run SimulationX models on the motion platform at Dresden University and is described in chapter five. It allows the evaluation of different machine configurations and their parameters within an interactive simulation. The paper concludes with a discussion of the advantages, drawbacks and possible solutions when using FMI from a user's perspective.

2 The FMI Standard

FMI, short for Functional Mockup Interface, is a new standard for model exchange [4] as well as co-simulation defined within the MODELISAR project. It describes a set of functions and their parameters which shall be part of a binary .dll (Dynamic Link Library) file or .so (Shared Object) file under Windows or Linux respectively. The binary file is complemented by an XML file which includes a description of the model and all its variables. The separation of model and description data leads to very efficient code, which may even be run on embedded systems, while maintaining usability for more powerful systems by offering the complementary XML documentation. Using a simple binary format has the advantage that it can be included at minimal effort in almost any application that allows incorporating user code. Thus, the FMI standard fills the gap between abstract Modelica models and vendor dependent co-simulation.

3 Initial Situation at Dresden University

One research topic of the Chair of Construction Machines and Conveying Technology at Dresden University is concerned with studying and optimising Heavy Mobile Machinery and its components by using interactive simulation methods. Therefore a modular software framework called SARTURIS has been developed [2]. It allows real time simulation of technical systems in a virtual reality (VR) environment and features easy and flexible hardware integration in addition to a powerful visualisation. SARTURIS is based on C++, uses freely available libraries and is platform independent. The software framework provides a diversity of applications for models of technical systems. It is suited for both the analysis of machine behaviour on a PC or laptop as well as an interactive simulation on a motion platform (Figure 1).

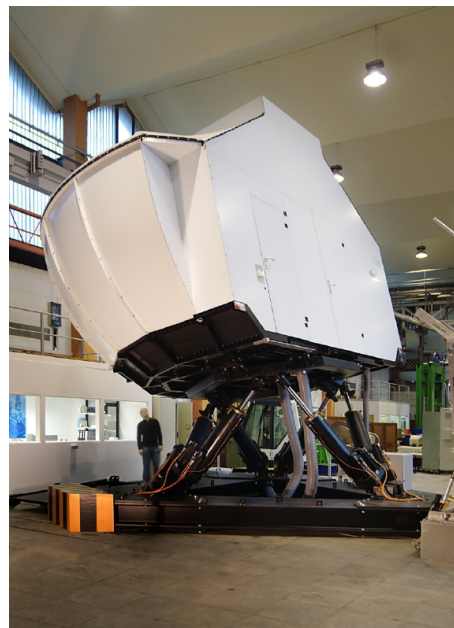


Figure 1: Motion Platform at Dresden University

In order to create simulation models, to run task-specific simulations or to evaluate simulation results different custom-designed software tools are used. Thereby the interaction of two or more tools can be useful and an exchange of data between them is necessary.

Figure 2 depicts the initial situation. Five basic proprietary exchange algorithms already existed or have been implemented for specific tools or tool combinations, providing the basis for different tasks:

1. Simulation tools based on Modelica for instance, are able to exchange models in general.
2. Many commercial tools like SimulationX, Simpack and Dymola support co-simulation with MATLAB/Simulink.
3. Other tools offer a wide range of output formats like PyMbs [2], a software for the realtime simulation of multibody systems.
4. Pynacolada, a software that supports recurring analysing tasks, features a broad range of input formats like MATLAB, SARTURIS and general HDF5 [5] data sets.
5. Specialised auxiliary tools or wrappers had to be developed to accomplish the exchange of required model data from OpenModelica and Simpack for an interactive simulation with SARTURIS.

Such tool couplings are important since experience has shown that general simulation environments are not always the best option for specialised engineering tasks. Next to classical simulations, models of technical systems can be used to solve other problems like inverse kinematics or finding optimal constructive geometries. Software designed for such tasks can only produce reliable results if the technical

system regarding the problem is described accurately enough. Those specialised tools however are rarely convenient modelling environments. If one can establish a comfortable and efficient data and model exchange, both software strategies could be combined with their advantages. Thus an expanded range of applications could be achieved.

Individual solutions for coupling tools (see option 5) have the great disadvantage that they are very specific and need constant attention. Extending the coupling to another tool usually calls for a different specialised data exchange strategy. This leads to the circumstance that not all possible exchange paths have been realised although many other tool interactions would be beneficial. The new FMI standard is a promising approach to provide a common foundation for universal model exchange and thus increasing flexibility while reducing maintenance efforts.

4 Integration of FMI

The introduction of FMI as the common interface between all tools led to a great simplification, see Figure 3. Now every tool simply exports FMI, imports it or supports both. Thus one single model can be made available to other tools very easily. There is no need for specialized utilities or wrappers anymore to establish data or model exchange between two tools. As soon as a tool features an FMI import or export it can be incorporated in every tool chain without any extra effort. Current versions of Modelica tools, such as SimulationX, JModelica and Dymola already feature FMI exports as well as FMI imports. For the Open Source alternative OpenModelica, the FMI export is currently being developed in a cooperation between Linköping and Dresden University and will be available in future releases. Outside the Modelica community FMI gains in

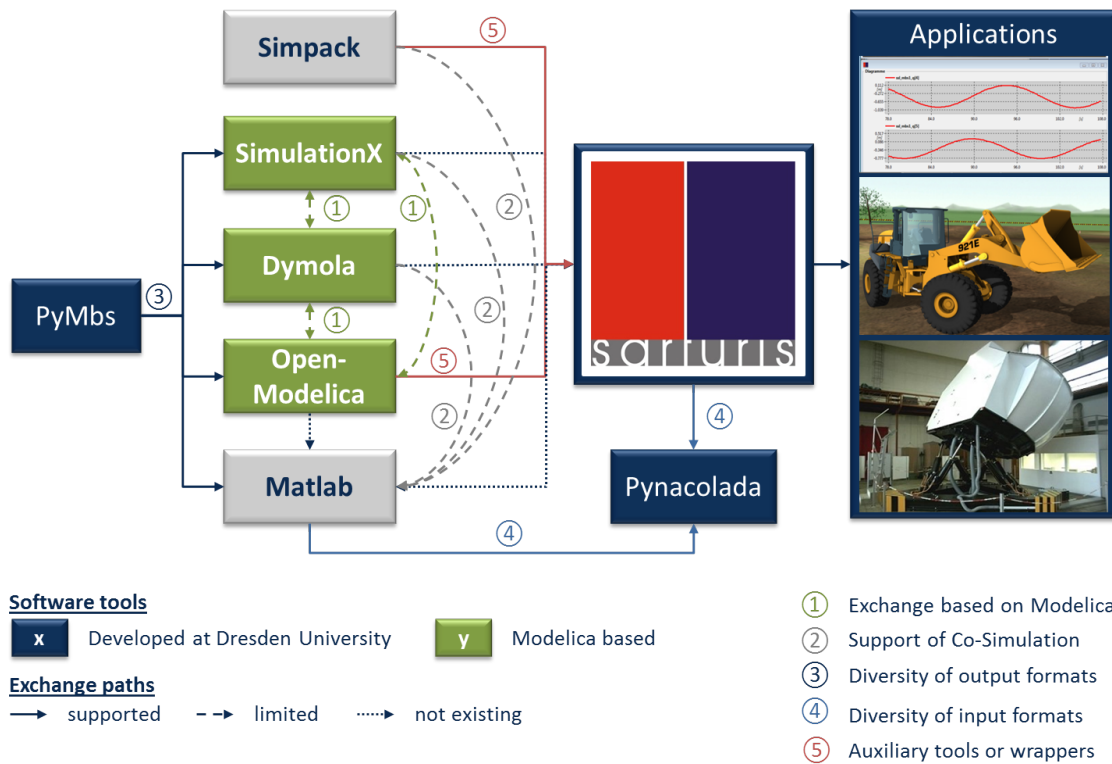


Figure 2: Tool Chains Before FMI

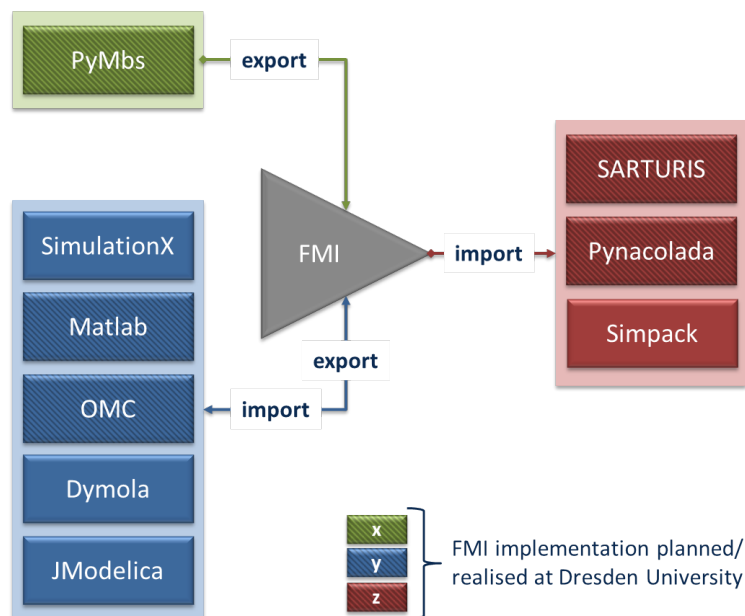


Figure 3: Tool Chains After FMI

importance as well. SIMPACK as a partner in the MODELISAR project for instance is developing an FMI import enabling SIMPACK users to include Hydraulics into their multi-body models using specialised software.

Unfortunately, no FMI implementation for MATLAB is known to the authors yet. Hence the authors started developing an import wrapper for both MATLAB and Simulink as well as an export for MATLAB Simulink models based on the C-Code generated by the Real-Time-Workshop (see <http://code.google.com/p/fmi2matlab/>).

Equipping self developed tools with FMI capabilities, proved to be a reasonable amount of effort. Due to the comprehensive documentation [4] as well as the FMU Software Development Kit provided by QTronic [6] an easy to understand reference implementation is available. Thus PyMbs, Pynacolada and SARTURIS do now support FMI.

5 SimulationX Models in Interactive Simulation via FMI

A particular interesting application of the new FMI based tool chain is the investigation of virtual prototypes using interactive simulation via the motion platform at Dresden University, see Figure 1. Whereas motion platforms are traditionally used to evaluate the influence of the machine and its complexity on the human operator, the aim of the Chair of Construction Machines and Conveying Technologies is the opposite. The motion platform is intended as a tool for studying the influence of the human operator on the machine. Thus it is possible to obtain joint forces and pressure distributions dependent on the behaviour of the operator and generally gain a deeper understanding of the machine and the ways to optimise it while it is in operation.

In order for the results to be significant a detailed model comprising mechanis, hydraulics, drivetrain and control systems has to be simulated. A way of developing such models has been presented by the authors in [2]. Due to the newly implemented FMI functionalities of SARTURIS one can now use SimulationX as well to set up models for the motion platform. Thus the full range of the SimulationX model libraries can be used which can considerably shorten the modeling effort.

To test the FMI based tool chain a model of a wheel loader, see Figure 4, has been developed together with ITI GmbH. In addition to the multibody system it consists of hydraulic cylinders and valves, a reduced drivetrain and Paceijka tire models. Furthermore it considers contact between the bucket and the underground. The model has been developed and thoroughly tested within SimulationX and it is capable of running in realtime.

As the new SimulationX offers FMI Export, the model can now be easily transferred to SARTURIS which in turn controls the motion platform. Within SimulationX all inputs which should be provided by the operator are defined as well as all outputs needed for visualisation and the evaluation of the model. The wheel-loader modelled with SimulationX is now running within an interactive simulation on the motion platform and can be driven through a virtual environment.

Along with the described benefits, there are problems that had to be overcome and shall be discussed in the following.

6 Visualisation

Although importing the model via FMI is easy to accomplish, the visualisation of the technical system has to be set up twice. Once in the modeling software for assembling the multi-body system and a second time in the simu-

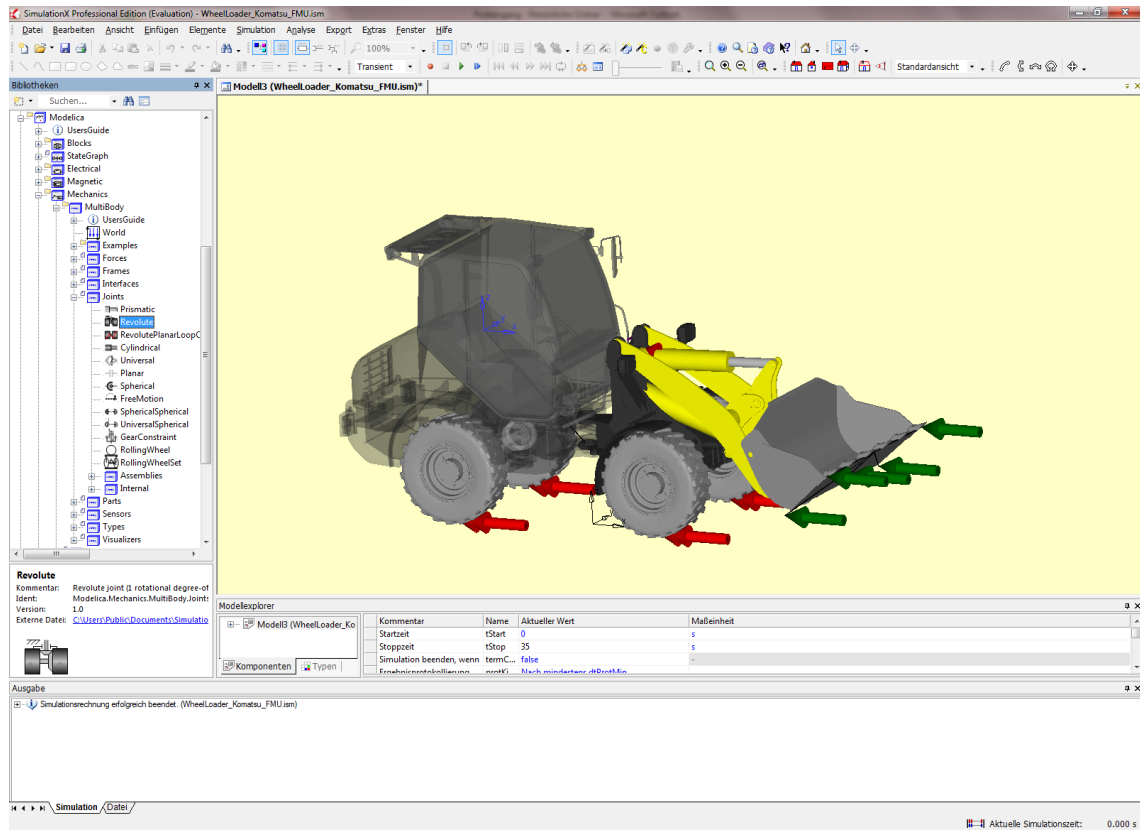


Figure 4: Model of a Wheel Loader in SimulationX

lation software for visualising the results. FMI does not incorporate any additional data concerning the graphics. According to the FMI philosophy, where the binary file is supposed to only contain the vital model information so that it can be run on embedded platforms, these additional information should not be stored in the code but either within the accompanying model description or within an extra file. Possible approaches might involve usage of the standards X3D [7] or COLLADA [8].

7 External Functions

7.1 Problem

It is often required to use the same vehicle model for different simulation tasks on different test tracks. Every vehicle model is therefore stored in a separate FMU. When performing contact detection, as needed for determining the tire contact point or general collisions, an extensive communication between the model and the track is needed.

Using FMI this can either be achieved by using inputs and outputs or linking user defined code statically. Statically linked user defined code is unsuitable since it would be difficult to communicate with the simulation environment. Even if it could be accomplished, the resulting FMU can only be used in combination with that particular environment. Using inputs and outputs are not suitable as well since their number may grow very large the more often a specific function is used. Also iterative methods would be hard to implement.

7.2 Possible Solution

What is needed is a coupling of the FMU and the simulation environment at runtime preferably via callbacks. Unfortunately, the current FMI interface does not support such a feature.

That is why for our models we extended the FMI interface by the following function

```
fmiStatus fmiSetExternalFunction(
    fmiComponent c,
    valueReference *vr,
    void *functionPointer);
```

passing a void function pointer to an *fmiComponent*. Therefore the accompanying modeldescription XML has been extended by all external functions, that can be set at runtime with their *valueReference* as well as their input and output parameters. Thus almost no additional code has to be included into the model itself.

7.3 Compatibility

Such an FMU may still be compatible to general purpose simulation tools. When instantiating an FMI component, all external functions and their signature are known from the modeldescription XML file. Thus, a user may choose an external function from all functions that are known to the tool and have the same signature.

7.4 Modelica

In order to create such an FMU from a Modelica tool, these functions have to be marked. This can either be done by introducing a new annotation or by extending the Modelica language. In fact, the Modelica standard 3.2 already offers *Functional Input Arguments To Functions* ([9], section 12.4.2). If this feature was to be carried over from *functions* to general *models* a clear description had been found.

7.5 Example

In order to illustrate this idea, a simple example shall be given. Consider the bouncing ball example:

```
model BouncingBall
    constant Real g = 9.81;
    parameter Real c = 0.9;
```

```

parameter Real radius = 0.1;
Real height(start = 1);
Real velocity(start = 0);
input HeightFunction heightFun;
equation
  der(height) = velocity;
  der(velocity) = -g;
  when height <= (radius + heightFun(time)
    ) then
    reinit(velocity, -c*pre(velocity));
  end when;
end BouncingBall;

```

The type HeightFunction may be defined as

```

partial function HeightFunction
  input Real t;
  output Real height;
end HeightFunction;

```

returning the height of the underground at a given time t.

A complete test model might look like

```

function SinusHeight
  extends HeightFunction;
algorithm
  height := sin(t);
end SinusHeight;

model SinBounce
  BouncingBall ball;
equation
  ball.heightFun = SinusHeight;
end SinBounce;

```

If the model BouncingBall were translated into an FMU the function pointer heightFun could be treated like any other component mapping it to void* and including it into the modeldescription.xml. In addition the xml description should include the signature of the function (inputs and outputs) as well as its comment which should be used as a short description what this function does.

8 Model Exchange Across Company Borders

One major advantage of FMI is that the model exchange described in this paper does not need to be limited to one institution or company only. Using FMI might also be a valuable

tool for cooperation between OEMs and their suppliers. Since it is virtually impossible to deduct information from models in binary format, suppliers can safely provide models to OEMs without revealing too much of their knowledge. However, in order to exchange a model in binary format only, all target platforms have to be known in advance. True platform independence is achieved by exchanging source code allowing more insight into the model.

9 Conclusion

It has been shown that powerful modeling tools are not always the best option for specialised simulation tasks. However, software designed for special tasks can only be as good as their underlying models of the technical system. Thus it is beneficial to exchange models between such tools.

Before FMI it has always been a very specialised solution which was hard to maintain. Using FMI at Dresden University has helped to unify the modeling and simulation software environment. It reduced the need for auxiliary tools establishing the model exchange and the time spent on developing and maintaining them. A major benefit is the option to run multidomain models created in SimulationX within an interactive simulation on the motion platform at Dresden University which can be used to study the influence of the operator on the machine and its components. Open problems like visualisation and calling external functions have been addressed as well as political advantages when using FMI.

References

- [1] Beater, P.; Otter, M.: Multi-Domain Simulation: Mechanics and Hydraulics of an

Excavator, Proceedings of the 3rd International Modelica Conference, November 2003, pp. 331-340

- [2] Frenkel, J.; Schubert, C.; Guenther, K.: Using Modelica for Interactive Simulations of Technical Systems in a Virtual Reality Environment, Proceedings of the 7th International Modelica Conference, Como, September 2009
- [3] MODELISAR consortium: MODELISAR Project Profile, http://www.itea2.org/public/project_leaflets/MODELISAR_profile_oct-08.pdf
- [4] MODELISAR consortium: Functional Mock-up Interface for Model Exchange, http://modelisar.org/specifications/FMI_for_ModelExchange_v1.0.pdf
- [5] The HDF Group: HDF5, <http://www.hdfgroup.org/HDF5/>
- [6] QTronic GmbH: FMU SDK 1.0.1, <http://www.qtronic.de/en/fmusdk.html>
- [7] web3D Consortium: X3D Specification, <http://www.web3d.org/x3d/specifications/>
- [8] Khronos Group: COLLADA - Digital Asset Exchange Schema for Interactive 3D, <http://www.khronos.org/collada/>
- [9] Modelica Association: Modelica - A Unified Object-Oriented Language for Physical Systems Modeling - Language Specification Version 3.2, March 2010

Modeling of Coal-Fired Power Units with ThermoPower Focussing on Start-Up Process

Sebastian Meinke¹ Friedrich Gottelt² Martin Müller¹ Egon Hassel¹

¹ University of Rostock, Chair of Technical Thermodynamics
A.-Einstein-Str. 2, 18059 Rostock, Germany

²XRG Simulation, Harburger Schlossstr. 6-12, 21079 Hamburg, Germany
sebastian.meinke@uni-rostock.de gottelt@xrg-simulation.de

Abstract

Governmental encouragement of renewable energies like wind energy led to an extensive increase of installed wind energy capacity worldwide. In order to allow a complete integration of this continuously fluctuating energy source, it is necessary to have a highly flexible operation management as well as power stations which are able to follow the high dynamics of the wind power production.

In this context the component fatigue and operational limitations of current and future power stations have to be investigated under the influence of enhanced plant dynamics.

For this purpose a detailed Modelica model of the hard coal fired steam power plant of Rostock, Germany is presented and extensively validated. The model makes use of the well-known non-commercial library ThermoPower. This Modelica Library is extended by models for common solid fuel burners and radiation-dominated firing zones. In addition to this, different approaches for modeling two-phase containers like the feed water tank are discussed. The derived model is used to compare different operation modes with respect to the occurring component wear.

Keywords: ThermoPower, coal fired power plants, firing modeling, two phase tank, power unit start up

1 Introduction

In a future power grid with high renewable power feed, especially from wind power, it becomes more important as well as economically beneficial for conventional power plants to be able to adjust the production in order to balance the renewable energies. But due to the long life time, the majority of current power plants have been designed decades ago mainly for steady

state operation. Consequently, the focus was put more on reliability and preservative operation than on high dynamics.

The recent and ongoing changes in the energy market in Germany will lead to an increased number of start-ups and load changes, which cause additional life time consumption. Improvements of the existing technologies are required to enable higher dynamics at limited additional stress during transient operation.

This is especially true for coal fired power plants because of the fuel pulverization in coal mills. These mills have a slow and often unknown dynamic and limit the load gradient of coal fired units. Additionally, the boiler itself shows a slow transient response due to its big metal and water masses as well as uncertainties like degradation of the heat transfer due to ash build up on the heating surfaces. To overcome this, the load change rates are made sufficiently slow. Improvements to this conservative approach could be achieved by the use of advanced control systems, e.g. state observers and model based control systems or additional sensors, like for example coal dust measurement [1].

For the evaluation of such optimizations of the process and the control system, computer aided simulation of the power plant process could be a powerful tool.

2 Scope of Investigations and Thermodynamical Model

In order to judge the expected impacts of a more dynamic power plant operation a detailed, transient model consisting of one-dimensional or lumped interlinked sub models, based on thermodynamic fundamental equations, is presented. The 550 MW hard coal power plant Rostock, that started its operation in 1994, has been used as a reference. The power plant repre-

sents the state of the art and is due to its long rest life time heavily effected by future changes of the energy market.

2.1 Object of Investigation

The power plant Rostock has a conventional, hard coal fired steam generator. This is a single direction once-through forced-flow boiler in Benson design, which is run in modified sliding pressure operation. The boiler is equipped with four superheater and two reheater heating surfaces. The fuel supply is carried out by a coal dust firing with direct injection. The combustion itself takes place in 16 NO_x -lean vortex staged burners, which are distributed on 4 burner levels with each 2 burners on the front and back side.

The main characteristics are provided in the following table (1):

Table 1: Key data of power unit Rostock, Germany [2]

power unit data	gross electric power net efficiency rate district heating max. degree of utilization	550 MW 43.2 % 300 MJ/s 62 %
boiler	manufacturer design life steam production SH-pressure/ -temperature RH-pressure/ -temperature firing coal mills	Babcock once-through forced-flow boiler single direction design single reheating 417 kg/s 262 bar/545 °C 53 bar/562 °C opposed firing, 4 levels combined coal dust/oil 4 roll wheel coal mill MPS 225, hard coal
turbo gen set	manufacturer design number of housings operation mode	ABB without regulating wheel 1 HP, 1 IP, 2 LP natural/modified sliding pressure

2.2 Overview of the Power Plant Model

Base for this power plant model is the non-commercial Modelica library ThermoPower [3]. Many of the comprised sub models in the ThermoPower library, like pipes, valves, metal walls, mixers, etc. have been used in this work or have been taken as a starting point for self-developed models. One of the newly added models is a generic two phase tank, that can be used for a feed water tank, preheaters, a start bottle or a condenser. Furthermore, models for cyclone separators, a combustion chambers and segments of a flue gas duct have been developed. Two of these new sub models are presented in the following section 3.

The focus of the investigation has been put on the water-/steam circuit, the combustion chamber of the steam generator and the fresh air passage within the coal mills, as well as their dynamics and the influence

of different operation modes on distinct devices e.g. thick-walled headers and turbine shafts.

A simplified schematic of the model is shown in figure 1. Indicated are the feed water pumps, high pressure preheaters (HPP), the steam generator, the different turbine stages, as well as the forced draft and mill fan, the air preheater and the coal mills. The low pressure preheaters (LPP) are not part of the power plant model, since they are not highly stressed, due to their low temperature level.

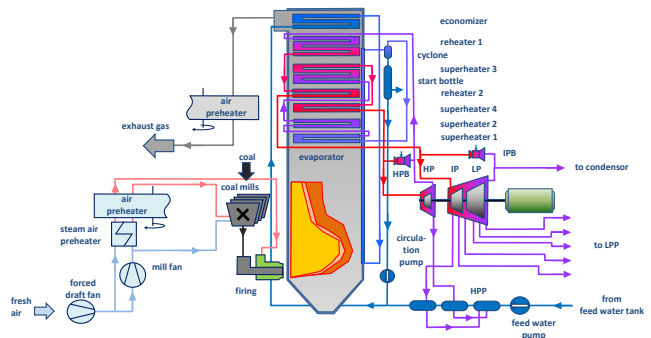


Figure 1: structure of power plant model

For making simulation-based statements about the influence of different power plant operation modes the thermodynamical model is coupled to a reduced copy of the power plant control system, which is implemented using the Modelica Standard Library components.

The implemented control system uses the currently calculated physical values (i.e. live steam parameter, generated power at a specific coal input) and in a consequence adjusts set values (e.g. live steam pressure) and manipulated variables (e.g. rotational speed of the feed water pump) of the water-steam cycle. Because of this feedback the grade of details as well as the accuracy of the modelled steam cycle and its interfaces to the control system needs to be reasonably high.

In detail the power plant control system sets the manipulated values using a map based pilot control. The expected control variable is predicted by a transfer function based model of the process. The difference between this predictive value and the corresponding measurement is adjusted via a corrective control loop, as described in the VDI/VDE guideline 3508 [7].

2.3 Level of Detail

In the following the degree of detail is explained by using the detailed reproduced boiler as an example. The boiler model differentiates eight separate heating surfaces: economizer, evaporator, four superheaters and

two reheaters. Between the superheaters SH1 and SH2 as well as SH3 and SH4 plus in between the reheaters RH1 and RH2, spray atomizers are located for live steam temperature control.

In compliance with the modular approach of Modelica any of the boiler's eight different heat exchangers is composed of different base models, see figure 2. Starting from highest temperature components the following modules can be found: The flue channel segment which models the energy storage as well as the gas side heat transfer due to convection and radiation, a sub model, that calculates the conductive heat transfer inside the metal wall of the pipes and a third module for the convective heat transfer occurring at the inner wall as well as a one-dimensional pipe flow model. The heat flow at the system boundary between these modules is implemented using connectors. On the gas side very complex heat and mass transfer conditions occur defined by a large range of temperature and a variety of geometric characteristics. To cope with this complexity semi-empirical heat transfer correlations for the different stages of the combustion chamber can be distinguished while the three-dimensional flow field is reduced to one dimension. The latter reduction neglects any deviations from the perfect symmetric temperature and flow field but is in congruence with the "one-pipe"-approach of the water and steam side of the boiler.

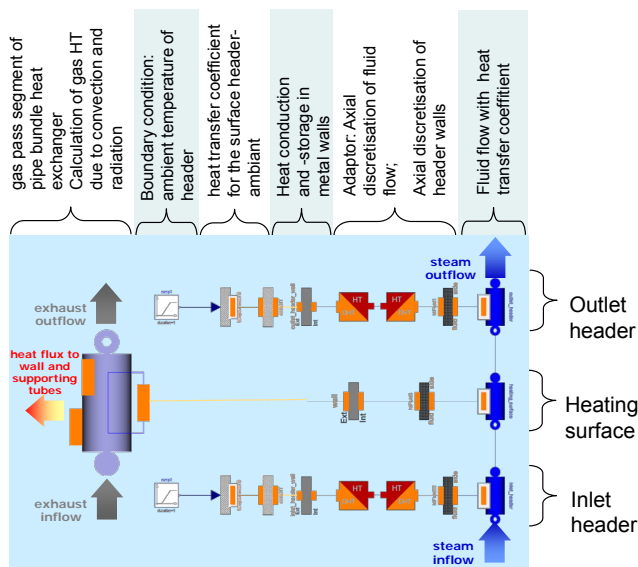


Figure 2: Dymola view of a generic superheater, which distinguishes inlet and outlet header and heating surface

Before and after each of the bundle heat exchangers a thick-walled header is located to allow a mixing

of several legs connected in parallel. Again, we find models for convective and conductive heat transfer and energy storage in the metal masses of the surrounding walls. In addition, there are adapter modules to couple components of different dimensionality. This is necessary because the headers are discretized in flow direction while the thick header walls are discretized in heat flow direction which is perpendicular to the first mentioned.

3 Additional Components

In the following chapter a selection of the developed components are presented, section 3.1 explains the sub models for the reproduction of the firing process and gas side heat transfer. In section 3.2 the modeling of a two phase tank is discussed.

3.1 Modelling of the steam generators gas side

Due to the complex, unsteady and 3-dimensional nature of a real firing process, a model of the combustion process, which is a part of an overall power plant model, needs to be strongly simplified. For that reason instant combustion is assumed and subsequently a lumped gas volume model is used for the reproduction of the combustion chamber, which works according to the principle of a homogeneous agitating tub [6] with uniform flue gas conditions (see figure 3).

The main input values for combustion calculation are fuel and fresh air properties. The description of the fuel is conducted by a raw coal composition and the coal mass flow, which is delivered by a coal mill model. The modeling of the coal mills has been done according to work of Niemczyk *et al.*, 2009 [4].

The raw coal composition can be obtained by an elementary analysis and integrates a single ash and embedded water fraction (equation 1). On the gas side the ThermoPower component SourceW was used for generating a combustion air mass flow. To indicate the preheated air conditions the moist air media of Modelica.Media has been applied. The air properties are functions of the boiler load.

The essential of this model is a simple combustion calculation without pollutants. Also implemented are functions for lower heating value according (see equation 2 [5]) and air ratio in dependency of desired firing power.

Other parameters are the ash fraction, which is suspended in the flue gas and the unburnt carbon fraction

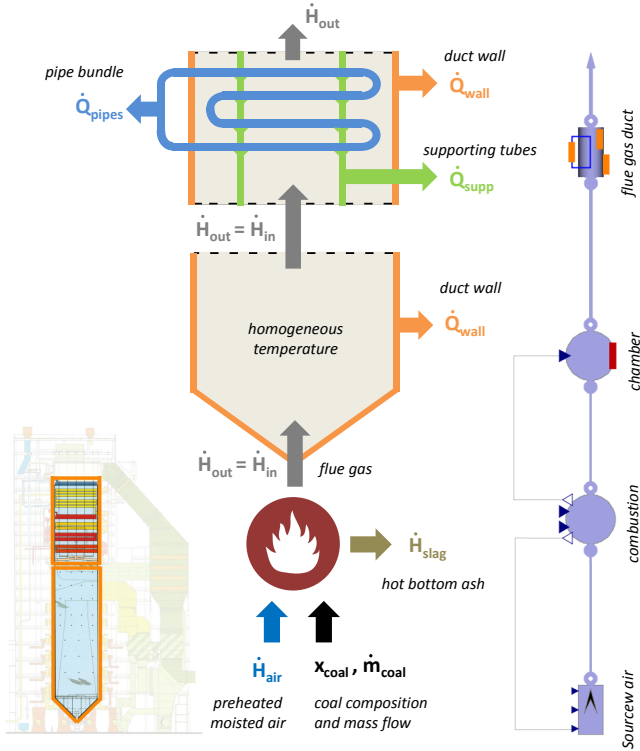


Figure 3: scheme of the combustion chamber and flue gas duct

that remains in the solid ash. This data is important for the particle radiation and is set to default values of 0.90 and 0.02, respectively. In result of the known energy balance the combustion model creates a flue gas with its adiabatic combustion temperature. Therefore the "simple flue gas" media model was introduced and its properties can be transferred to an output flow connector. (see equation 3).

$$x_C + x_H + x_S + x_O + x_N + x_W + x_A = 1 \quad (1)$$

$$H_u = 34.8(x_C - x_{Cu}) + 93.8x_H + 10.46x_S + 6.28x_N - 10.8x_O - 2.45x_W \quad (2)$$

$$x_{i,RG} = \frac{\hat{m}_{i,RG}}{\hat{m}_{RG}} \quad (3)$$

The heat transfer from the flue gas to the different surface areas is caused by two physical mechanisms, radiation and convection. Convection is heat transfer via particle transport and works on all overflowed surfaces. The formula 4 uses a heat transfer coefficient, which depends on the fluid properties, the flow velocity and the geometry of the surface (see equation 5). Every vertical surface (membrane duct wall and

the supporting tubes, which are carrying the pipe bundles) is handled like an overflowed plate. The Nusselt number is calculated according to [6].

The streaming around the pipe bundle surface is a forced flow across a tube. In this case the Nusselt number calculation is extended by a geometrical alignment factor [6]. The used flue gas properties are averaged values between two nodes. For ribbed tubes (economizer) an effective heat transfer coefficient will be computed according to [5], which takes the rib effects into consideration.

$$\dot{Q}_{conv} = \alpha_{conv} \cdot A_{wall} \cdot (T_{gas} - T_{wall}) \quad (4)$$

$$\alpha_{konv} = \frac{Nu \cdot \lambda}{l} \quad (5)$$

Especially in higher temperature regions above 1000 °C (combustion chamber) the heat radiation is the dominant heat transfer mechanism (see equation 6). The radiation sources are the flue gas, the contained solid particles (dust) and the surfaces. The influences of these three sources are expressed in different dimension free emission (ϵ) and absorption (a) coefficients.

In the flue gas only the components H_2O and CO_2 are relevant emitters. The combined emission coefficient of both is calculated using equation 7 [6]. The calculation of the emission coefficients for dust (in fact unburnt carbon and flying ash) is also described in [6]. The implemented functions also consider the back radiation from the wall to the flue gas represented by the absorption coefficients.

$$\dot{Q}_{rad} = \epsilon_{gas/wall} \cdot \sigma \cdot A \cdot (\epsilon_{gas/dust} \cdot T_{gas}^4 - a_{gas/dust} \cdot T_{wall}^4) \quad (6)$$

$$\epsilon_{gas/wall} = \frac{\epsilon_{wall}}{1 - (1 - a_{gas/dust}) \cdot (1 - \epsilon_{wall})} \quad (7)$$

Considering the discharged heat flow to the membrane wall the homogeneous temperature in the combustion chamber is calculated by the energy balance. The main parameter for adjustment is the introduced fouling correction factor. This factor is implemented into the energy balance and represents the fouling of the heat surfaces. The fouling decreases with lower flue gas temperatures.

Subsequently the exhaust gas is flowing to the flue gas duct, which is the second part of the steam generator and located above the combustion chamber. For effective modeling it is divided in segments - one for

every pipe bundle heat exchanger. Each sub model is based on the 1-dimensional gas flow model of the ThermoPower library. The modified models have three heat transfer ports - one for the flue gas duct wall (evaporator), the supporting tubes (superheater SH 1) and the pipe bundle itself (superheaters, reheater and economizer) - shown in figure 4.

An exception is the superheater SH 1, its pipe bundle is at the lower end of the supporting tubes and the heat transfer will be computed by empiric equations [5]. For every flue section following parameters can be defined: number of nodes, geometry and fouling factor. In cooperation with the other models of the ThermoPower library it is easy to construct direct- or counter-current heat exchangers.

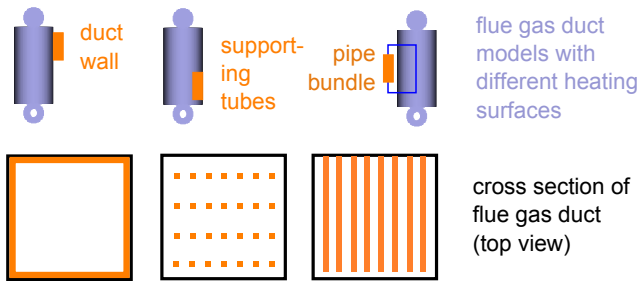


Figure 4: overview of the heating surfaces of the flue gas duct

3.2 Two-Phase Tank

In order to decouple the low pressure part from the high pressure part of the water-steam cycle a large storage tank is located between the low pressure preheaters and the feed water pump. This tank typically contains both, steam and liquid water. The two phases interlink to each other by heat and mass transfer. During slow load changes and steady state operation water and steam will be in an equilibrium, i.e. the phases will be near the dew and boiling curve, respectively. Due to limited heat transfer between both phases a constant temperature difference between the phases will arise. In contrast to this, during fast state changes, as they may occur during condensate hold-up, a significant fraction of steam may be present in liquid phase and vice versa. To cope with this effects a model solving energy and mass balance for both phases separately was implemented, using the central equations below. Herein the generic inlet and outlet ports *in* and *out* are defined as ThermoPower components flange :

```
//mass balance: Index v = vapor
//                Index l = liquid
der(Mv) = in.w*x_in + out.w*x_out + w_ev
- w_con;
der(Ml) = in.w*(1-x_in) +
out.w*(1-x_out)- w_ev + w_con;
// energy balance vapor phase
der(Hv) - der(p*Vv) =
(in.w*x_in+wzero)*h_in_v +
(out.w*x_out+wzero)*h_out_v - Qflow +
(w_ev+wzero)*hvs - (w_con+wzero)*hls;
// energy balance liquid phase
der(Hl) - der(p*Vl) =
(in.w*(1-x_in)+wzero)*h_in_l +
(out.w*(1-x_out)+wzero)*h_out_l + Qflow -
(w_ev+wzero)*hvs + (w_con+wzero)*hls;
```

Thereby linking the two phases via the evaporation and condensation mass flow rates and the heat exchange via the common surface:

$$w_{ev} = \max(0, \tau \cdot (h_l - h_{ls}) / (h_{vs} - h_{ls}) \cdot M_l);$$

$$w_{con} = \max(0, \tau \cdot (h_{vs} - h_v) / (h_{vs} - h_{ls}) \cdot M_v);$$

$$Q_{flow} = A_{sup} \cdot \alpha \cdot (T_v - T_l);$$

The code omits the heat exchange with the surrounding walls for the sake of simplicity. However, the model makes use of some unknown parameters, namely τ as the time constant for phase change processes and α as the heat transfer coefficient (HTC) at phase interface. For this reason a parameter variation was done to gain information about the influence of these parameters.

For evaluation a temporary reduction of the condensate mass flow entering the feed water tank is considered. Usually the condensate flow is controlled to keep the feed water tank level at a defined set value but when short-term generator power is needed the condensate mass flow is reduced thus increasing the turbine mass flow in an indirect manner. This condensate holdup is the state-of-the-art method for primary regulation. The scenario comprehends a reduced condensate mass flow rate over approximately 13 min starting at $t = 60$ min while a constant mass flow is extruded by the feed water pumps. In consequence the water level decreases, see figure 5. After finishing the condensate holdup the controller starts filling the container again by massively increasing the condensate inflow into the tank. Considering the variation of the heat transfer between the phase interface we find that the time response of the level is not affected by differing interface heat transfer coefficients. In contrast to this the tank pressure is qualitatively and quantitatively influenced by this parameter, see figure

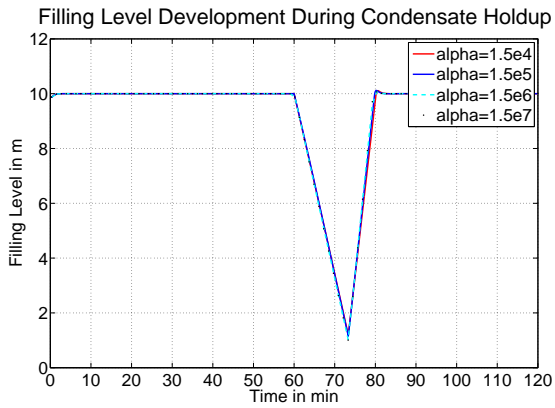


Figure 5: Tank Pressure development during condensate hold-up

6. The stationary tank pressure is lower for cases with near-ideal heat transfer indicated by high heat transfer coefficients $\alpha \geq 1.5 \cdot 10^7 \text{ W}/(\text{m}^2\text{K})$. The scenario with heavily reduced heat transfer leads to decreasing pressures during condensate holdup because the effect of the reduced cooling by the cold condensate is overcompensated by the decompressing effect of the emptying process. For the cases with a high energetic coupling of the two phases we find the cooling effect of the condensate to outweigh the emptying process. The time constants for phase change have a similar phase-coupling impact as the heat transfer but turns out to be quantitatively of minor influence and is therefore not discussed in detail.

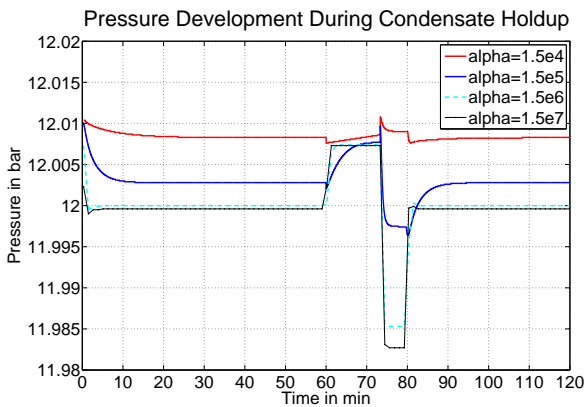


Figure 6: Tank Pressure development during condensate hold-up varying the HTC

The question arising from figure 6 is how intense the heat transfer between the liquid and gaseous phase is. Analysis of continuous measurement data provided by the power plant operator revealed a fairly constant temperature difference between the liquid temperature and the saturation temperature gathered from the tank

pressure $\Delta T \approx 3.7\text{K}$. This suggests a good heat transfer between the phases referring to a heat transfer coefficient of $\alpha = 1.5 \cdot 10^5 \text{ W}/(\text{m}^2\text{K})$. This high heat transfer coefficient, typical for good inter-phase mixing, is ensured by the applied Stork spray injector.

When considering the pressure development and the subcooling of the liquid water in the tank it becomes clear that the implementation of separate phase balancing has only minor influence on the global behavior of the component. Therefore, the model might be reduced using only one lumped state into account. This state will be in the wet steam region during proper operation of the tank, i.e. $0 \leq \text{level} \leq \text{level}_{max}$:

Mass balance for both phases:

$$\text{der}(M) = \text{in.m} + \text{out.m};$$

Energy balance for both phases:

$$\text{der}(H) - V \cdot \text{der}(p) = \text{in.w} \cdot h_{in} + \text{out.w} \cdot h_{out} ;$$

With boiling vapor at the outlet of the liquid phase interface:

$$h_{out} = h_{ls}$$

The reduced model shows - as stated - similar transient behavior concerning mass storage, see figure 7. In contrast to this, a different time development of the tank pressure has to be stated. The steady-state value is slightly lower for the simplified model and differs during transients. However, the deviation is below 1 % relative error for both transient time period and steady state and the pressure time derivatives show the same sign.

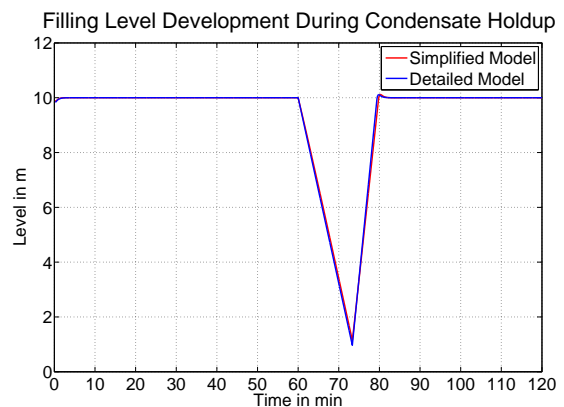


Figure 7: Filling level development during condensate hold-up applying different levels of detail

The effect of model simplification on the required condensate mass flow is shown in figure 9 and only small deviations must be stated. Thus, for controller design of the condensate pump the simplified model may be sufficient.

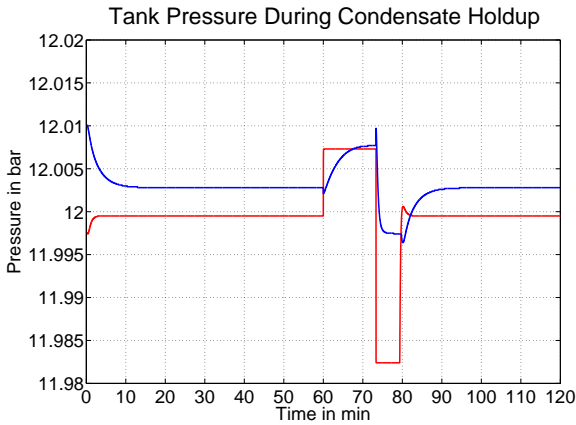


Figure 8: Tank pressure development during condensate hold-up applying different levels of detail

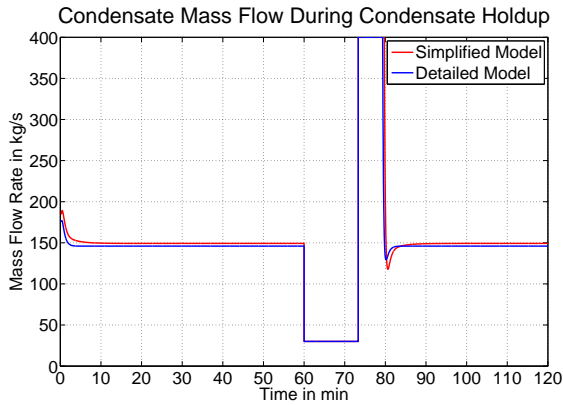


Figure 9: Condensate mass flow development during condensate hold-up applying different levels of detail

The mentioned deviations of the tank pressure for the detailed and simplified model may induce differing tapping mass flows. Tappings are controlled or fixed steam extractions providing steam from the turbines for heating of the low pressure preheaters and the feedwater tank. If the tapping valve is not controlled the corresponding mass flow rate is defined by the pressure states in the turbine and the low pressure path. Thus, the heating mass input may differ between both models requiring different condensate mass flows. In the consequence the effective turbine mass flow in the low pressure turbine stages is different when integrating the tank models in a complete steam cycle. Considering a conservative estimation in a scenario of 80 % relative load this results in approx. 2 % relative error of the generator power.

Comparing the models presented both, advantages and drawbacks, apply to each of the model approaches. The detailed model considering two different phases

promises better numerical stability and more realistic results, especially during short-term transients while the simplified model is easier to initialize and will need less computational effort. Testing the convenience of the models under different usage conditions with respect to initialization and simulation progress will be a subject item of future investigations.

4 Validation of the Model

In order to check the created model on correctness, comparative measurements have been recorded in the power plant Rostock, which are shown below in contrast to the results of the simulation.

In the investigated period of time the power plant begins operation after 37 hour shutdown period with two subsequent positive load changes up to 95 percent. The development of the desired net power shown in figure 10 is the input to the model and is processed by the control system to a desired firing power signal and subsequently transferred to the coal mills and oil burners. Despite the simplifications in the reproduced control system of the power plant model a good reproduction could be achieved, differences between the model and measurement could be caused by parameter changes of the process, especially during a start, like a drifting heating value, uncertainties in the mill dynamic or degradation of the heating surfaces.

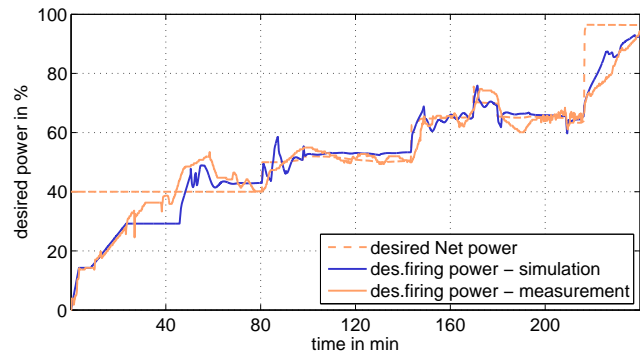


Figure 10: Definition of validation setup - power request from the load dispatcher and corresponding desired firing power

Beneath the power request the plant model needs two additional inputs, the state of the condensate water, which is provided by a load dependent table and the composition of the used coal. Since coal is not an homogeneous fuel the mass fraction of containing ash and water of the modeled coal are fluctuating in random intervals by +/- 1%. All other boundary conditions for the model, like furnace outlet and condenser

pressure as well as air inlet temperature have constant values.

Since the dynamic behavior of the overall process is mainly dominated by the fuel pulverization in the coal mills and the transient response of the boiler, it would be beneficial to validate the sub models for those two systems independently. But this is not possible, because the measurement of the system response of the coal mills, the coal dust mass flow rate upstream of the burners, is only possible with high efforts and low accuracy. Thus, such a measurement system is not standard and is not available. In a consequence, validation of the overall model is conducted by the use of the steam properties in the boiler and the generator output.

The live steam pressure and mass flow arise from the current heating by the furnace and the cooling from the working medium due to the feed by the feedwater pump. In figure 11 simulated and measured pressures at outlet of the boiler are compared. After 50 min the power plant is in sliding pressure operation and the life steam pressure is changing proportional with the load. In general a good conformity of both graphs can be stated. Variations (e.g. between 40 and 70 min) could be explained with differences in the firing power. Considering this, a good reproduction of the hydraulic characteristic can be stated.

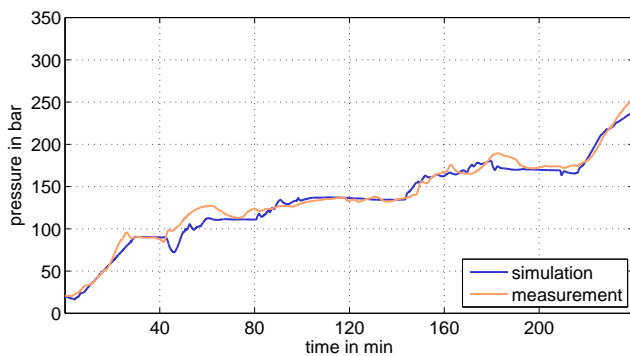


Figure 11: Comparison of calculated and measured life steam pressure at the boiler outlet

In figure 12 the development of the boiler inflow (Eco in) and outflow (life steam) mass flow rate is shown. In the first 50 min the boiler is in circulation mode and a minimum amount of 143 kg/s of feedwater is entering the economizer and the unevaporated water fraction is separated after the evaporator and looped back to the boiler inlet. Then the power plant switches to Benson operation and the feed water as well as the life steam mass flow rate is proportional to the load.

In order to verify the proper representation of the

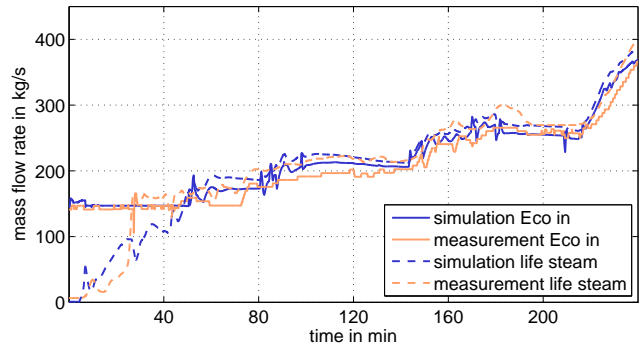


Figure 12: Comparison of calculated and measured economizer in and superheater out mass flow rate

heat transfer from the exhaust gas to the different heating surfaces by the model, the calculated steam temperatures at the outlet header of each heating device are compared to the corresponding measurement. In the following some representative plots are discussed. In figure 13 the water temperature entering the boiler at the economizer inlet and the steam temperature after the evaporator is brought out with respect to time.

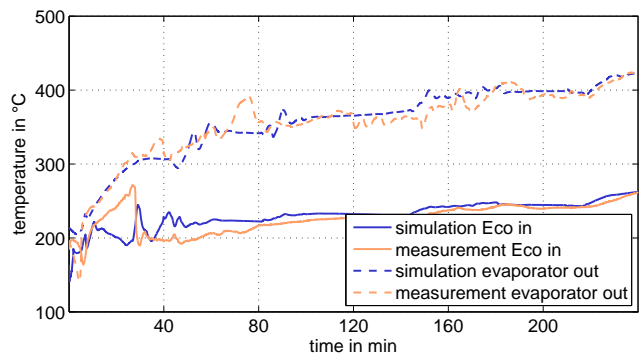


Figure 13: Comparison of calculated and measured fluid temperature at inlet of the economizer and after evaporator

Whereas the simulated evaporator outlet temperature shows a good agreement with the measurement - the mean relative error is about 3.3 %, the economizer inlet temperature shows a deviation between 10 and 30 min with a maximum error of 28 %. This is caused by a difference in the circulation rate: in the simulation a large quantity of cold feed water is lowering the eco inlet temperature in contrast to the measurement. This means that a higher amount of almost boiling circulated water, as can be seen in figure 12 as the difference of the eco in and life steam flow, is leading to a high eco inlet temperature. In Benson operation the error is less than 5.2 % and shows a very good correlation.

The trend of the temperature after superheater SH 1

shows analogue results, hence the furnace and heat transfer model is capable to reproduce the heat transfer of the boiler and its temporal behaviour (see figure 14).

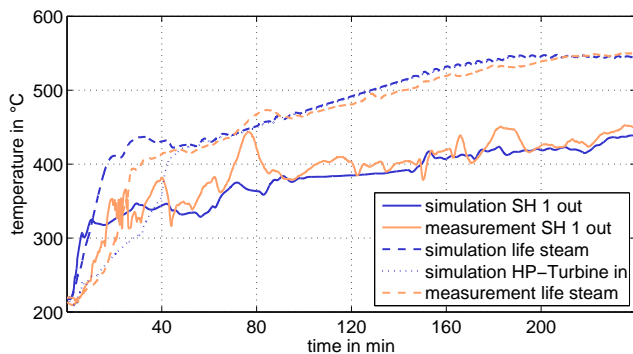


Figure 14: Comparison of calculated and measured steam temperatures after the first and the last superheater

In the same diagram the simulated and measured live steam temperatures are plotted. During the first phase of a start with low flow through the life steam pipes to the turbines, an inhomogeneous temperature distribution and the unknown location of the sensor complicates a validation, but the measurement is within the model temperatures at the in- and outlet of the life steam pipe.

During sliding pressure operation again a very high correlation between simulation and measurement can be stated. This can only be achieved by copying the cascaded PI-controller of the spray attemperators, that ensures the right tempering of the live steam.

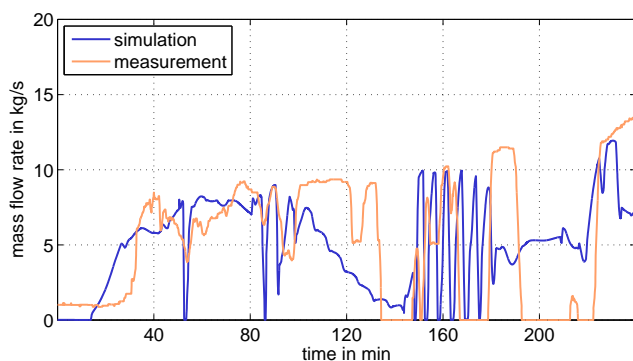


Figure 15: Comparison of calculated and measured injection mass flow rates at the first HP-injector (after first superheater)

Figure 15 shows the comparability of the implemented temperature control by comparing the simulated and measured injection mass flow rates. Both measured and simulated injection flow rates agree

qualitatively very well and show similar dynamics.

The figure shows that the attemperator is operated at its limits which leads to quite extreme changes of valve opening between fully opened and fully closed. This behavior is uncritical, since the task of the first injector is it to keep the second injector within its proper operation limits.

As can be seen in figure 16 the net generator power, whose behavior is effected by the dynamics of the entire power plant process, shows a good correlation. Thus, it can be stated, that the dominating and relevant effects are reproduced by the model and the validation shown above approves the accuracy and the validity of the model assumptions.

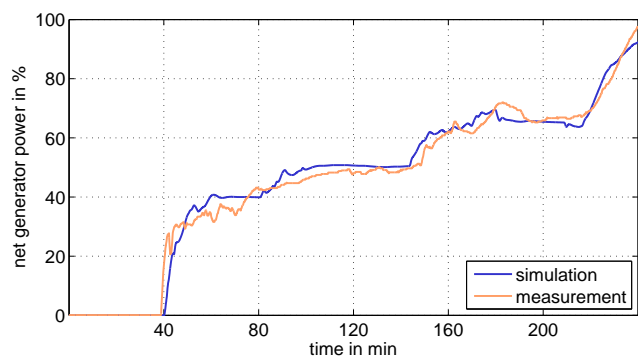


Figure 16: Comparison of the simulated net generator power and corresponding measurement

5 Evaluation of Component Strain

With this existing model it is possible to predict temperatures and temperature gradients at points which are inaccessible to measurements like wall temperatures of highly stressed components.

For the first 90 min of the presented soft start the occurring wall temperatures of the superheater outlet header are displayed in figure 17. Obviously the metal temperature at the outside of the wall follows the inner temperature with a certain delay and its amplitudes are considerably smaller. This effect can be explained with time specifics of the heat conduction and energy storage. The noticeable phase shift of the temperatures leads to relative high temperature differences between the inner and outer fiber in case of sharp edged changes in evaporator heating or cooling.

The evaluation of metal temperatures offers the possibility to benchmark different controller parameter sets with a view to preserving operation at concurrently high load dynamics.

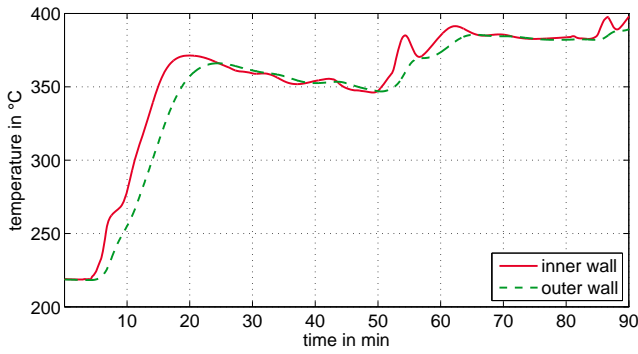


Figure 17: Metal temperatures in the superheater SH 2 outlet header

Quantification of the effects of thermal stress on the different components of a plant is a challenging task as the processes of fatigue are complex and highly statistical. For this reason the results of a fatigue prediction in this context can only be of qualitative nature and should be understood as a trend indicator that is capable of identifying the most stressed components and predict possible side effects of innovative control strategies on this complex system. For a detailed investigation of certain components a FEM-analysis should be carried out considering the installation situation (and with it possible pretensions in the component) and the exact geometry.

However, for a first estimation of the effects of more dynamic plant operation in the future two different approaches are used and should be discussed in the following:

The guidelines of the *Deutsche Dampfkesselausschuss* (2000) TRD 301 [8] and 508 [9] give directives for the estimation of fatigue of thick-walled boiler components under smouldering pressure and temperature due to start-up processes. For this purpose an effective stress range is evaluated with a Wöhler diagram for crack initiation. The following equation gives the law for calculating the stress range $\Delta\sigma$.

$$\Delta\sigma_i = \left(\alpha_m \frac{d_m}{2s_b} \right) \Delta p + \left(\alpha_\vartheta \frac{\beta_{L\vartheta} E_\vartheta}{1 - \nu} \right) \Delta\Theta \quad (8)$$

Herein α_m , α_ϑ , d_m , $\beta_{L\vartheta}$, E_ϑ , ν , Δp and $\Delta\Theta$ denote for mechanical and thermal correction factor for stress superelevation at branches, mean diameter, mean wall thickness, linear expansion coefficient, Young's modulus, Poisson's ratio and the range of pressure and temperature difference during load change, respectively. Figure 18 shows qualitatively the evaluation of the working stress during load change. The maximum number of load changes comparable to the actual one

is generated from the Wöhler-curve. The percentile fatigue of the actual load change is then:

$$e = \frac{1}{N} 100 \quad (9)$$

This estimation leads to conservative results in order to handle the numerous uncertainties in calculation of working stresses at complex components and material properties.

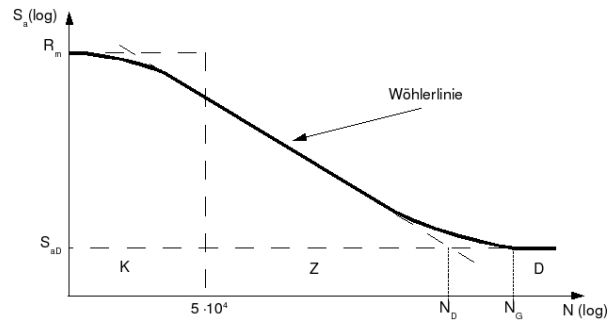


Figure 18: Principle of evaluation of component stress for cyclic loading [11].

This method allows to benchmark different operation modes in terms of their level of deterioration to different components. In figure 19 the fatigue of a soft start and several load changes is plotted for the inlet and outlet headers of the superheaters and reheaters. Please note that currently normal operation is between 50 % and 100 % load, so the shown load change of 60 % could be considered as an unconventional operation.

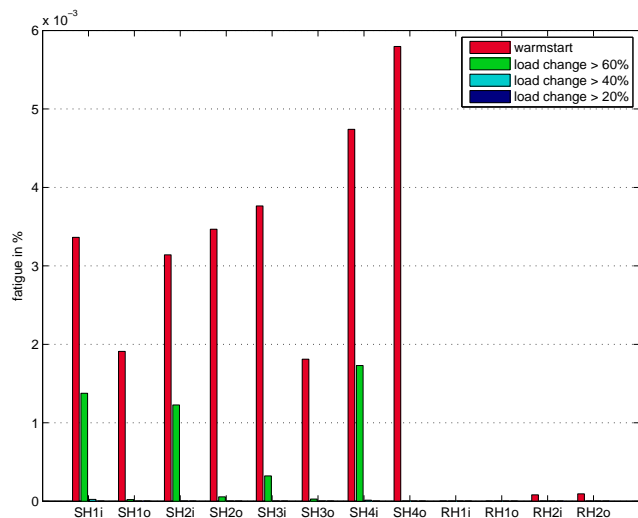


Figure 19: Fatigue of heating surface in- and outlet headers for different base stress situations

It can be obtained, that the outlet header of superheater SH 4 is effected the most, whereas the headers of the reheaters are not or lowly stressed. Furthermore it can be derived, that load changes less than 40 % barely cause any fatigue, because the stress levels are below the endurance strength.

Considering the flaw growth of predamaged component gives a far more sensitive view on the operation mode. The German *Forschungskuratorium Maschinenbau* [10] gives guidelines for the calculation of crack progress. Figure 20 gives a general overview on crack propagation rate as function of the range of stress intensity factor ΔK .

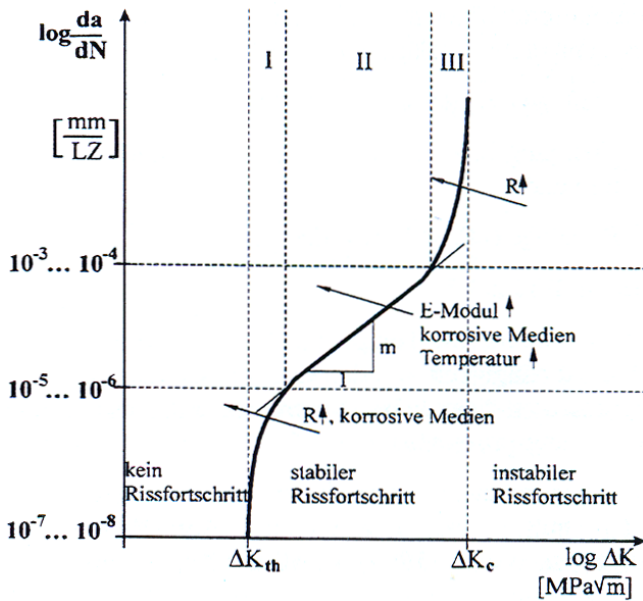


Figure 20: Overview on crack propagation under cyclic load [10].

According to figure 20 there is a certain load that does not leads to crack propagation ($\Delta K \leq \Delta K_{th}$). In region I to III there is a stable propagation to be expected ($\Delta K_{th} \leq \Delta K \leq \Delta K_c$) which can be conservatively estimated by the law of Paris and Erdogan:

$$\frac{da}{dN} = C\Delta K^m \quad (10)$$

Where a , N , C , m denotes for crack length, number of cycles, a case-specific factor and a load specific exponent, respectively. The stress intensity factor has to be calculated depending on the flaw's geometry and size and its position within the component. With this tool it is possible to detect the most strained components by comparing the crack growth over a certain reference time period.

In an analogue manner as in figure 19 the flaw propagation is shown for thick-walled headers in figure 21.

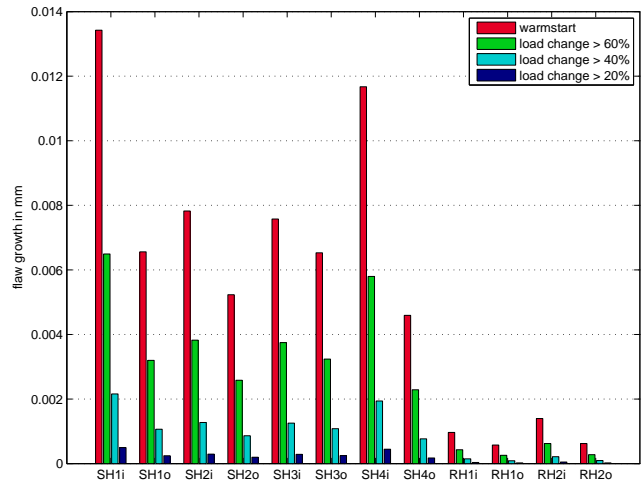


Figure 21: Flaw growth in potentially pre-damaged thick-walled in- and outlet headers for different base stress situations

In contrast to the fatigue also low stress levels of small load changes cause impairment and consequently with this estimation a method is given to evaluate the deterioration potential of normal operation.

6 Conclusion and Outlook

A detailed model of a coal-fired power unit has been implemented and extensively validated. The model makes use of the open-source Modelica library ThermoPower. A number of components especially for modeling of start-up-specific components, like the cyclone separator and the start-up bottle as well as models for the description of the fuel conversion and heat transfer in the firing of the plant have been implemented.

As an example for an application of the developed model some base operation scenarios, like load changes and a soft start, have been evaluated in terms of life time consumption. In a next step the influence of increased load gradients will be investigated.

The modular structure of the model allows the easy replacement of single components, e.g. life steam temperature control, which enables the benchmark of advanced control systems or the implementation of different or additional hardware for different operation scenarios. In this way, future demands on power plants, which might become necessary in order to realize wind integration successfully at controllable costs, can be benchmarked. This aspect of power plant operation management will probably become more important due to highly increasing wind power production

and its fluctuating characteristic.

The detailed manner of the plant model does not allow long term simulation over years or even weeks due to high computing time at this actual state of development. Therefore the fatigue has to be extrapolated in a first step assuming a constant or a repetitive operation mode for a long time period. Future work could cover a model reduction to increase its efficiency.

- [10] Forschungskuratorium Maschinenbau Bruchmechanischer Festigkeitsnachweis für Maschinenbauteile. VDMA-Verlag, 2001.
- [11] Lewin,G./ Lässig, G./ Woywode,N. Apparate und Behälter: Grundlagen Festigkeitsrechnung. Berlin, Verlag Technik, 1990.

References

- [1] Dahl-Soerensen, M.J./ Solberg, B. Pulverized Fuel Control using Biased Flow Measurements. In: IFAC Symposium on Power Plants and Power Systems Control, Tampere, 2009.
- [2] Hojczyk,B./ Hühne,W./ Thierfelder,H.G. Das Steinkohlekraftwerk Rostock. In: VGB Kraftwerkstechnik 77, 1997.
- [3] Casella,C./ Leva,A. Open Library for Power Plant Simulation: Design and Experimental Validation. In: proceedings of 3rd. International Modelica Conference, Linköping, 2003.
- [4] Niemczyk,P./ Andersen,P./ Bendtsen,J.D. /Soendergaard Pedersen,T. /Ravn,A.P. Derivation and validation of a coal mill model for control. In: IFAC Symposium on Power Plants and Power Systems Control, Tampere, 2009.
- [5] Effenberger Dampferzeugung. Berlin, Springer-Verlag, 2000.
- [6] Verein Deutscher Ingenieure VDI-Wärmeatlas, 10. Auflage. Berlin, Springer-Verlag, 2006.
- [7] VDI/VDE-Gesellschaft Mess- und Automatisierungstechnik VDI/VDE - Richtlinie 3508: Block-Führung/-Regelung von Wärmekraftwerken, 2002.
- [8] Deutscher Dampfkesselausschuss Technische Regeln für Dampfkessel (TRD) 301 Berechnung auf Wechselbeanspruchung durch schwellenden Innendruck bzw. durch kombinierte Innendruck- und Temperaturänderungen. Carl Heymanns Verlag KG, 2000.
- [9] Deutscher Dampfkesselausschuss Technische Regeln für Dampfkessel (TRD) 508 Zusätzliche Prüfungen an Bauteilen berechnet mit zeitabhängigen Festigkeitswerten. Carl Heymanns Verlag KG, 2000.

Dynamic modelling of a combined cycle power plant with ThermoSysPro

Baligh El Hefni

Daniel Bouskela

Grégory Lebreton

EDF R&D

6 quai Watier, 78401 Chatou Cedex, France

baligh.el-hefni@edf.fr

daniel.bouskela@edf.fr

gregory.lebreton@edf.fr

Abstract

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.

To that end, the library contains over 100 0D/1D model components such as heat exchangers, steam and gas turbines, compressors, pumps, drums, tanks, volumes, valves, pipes, furnaces, combustion chambers, etc. In particular, one and two-phase water/steam flow, as well as flue gases flow are handled.

The library has been validated against several test-cases belonging to all the main domains of power plant modeling, namely the nuclear, thermal, biomass and solar domains.

The paper describes first the structure library. Then the test-case belonging to the thermal domain is presented. It is the dynamic model of a combined cycle power plant, whose objective is to study a step variation load from 100% to 50% and a full gas turbine trip. The structure of the model, the parameterization data, the results of simulation runs and the difficulties encountered are presented.

Keywords: Modelica; thermal-hydraulics; combined cycle power plant; dynamic modeling; inverse problems

1 Introduction

Modelling and simulation play a key role in the design phase and performance optimization of complex energy processes. It is also expected that they will play a significant role in the future for power plant maintenance and operation. Regarding for instance plant maintenance, a new method has been devel-

oped to assess the performance degradation of steam generators because of tube support plate clogging, without having to wait for the yearly plant outage to open the steam generators for visual inspection [1].

The potential of Modelica as a means to efficiently describe thermodynamic models has been recognised for quite a while [2, 3] and lead to the initiative of developing a library for power plant modeling within the ITEA 2 EUROSYSLIB project.

This library is aimed at providing the most frequently used model components for the 0D-1D static and dynamic modelling of thermodynamic systems, mainly for power plants, but also for other types of energy systems such as industrial processes, energy conversion systems, buildings etc. It involves disciplines such as thermohydraulics, combustion, neutronics and solar radiation.

The ambition of the library is to cover all the phases of the plant lifecycle, from basic design to plant operation. This includes for instance system sizing, verification and validation of the instrumentation and control system, system diagnostics and plant monitoring. To that end, the library will be linked in the future to systems engineering via the modeling of systems properties, and to the process measurements via data reconciliation.

Several test-cases were developed to validate the library in order to cover the full spectrum of use-cases for power plant modeling:

- static and dynamic models of a biomass plant [8],
- dynamic model of a concentrated solar power plant,
- dynamic model of steam generators for sodium fast reactor [7],
- dynamic model of a 1300 MWe nuclear power plant covering the primary and secondary loops,
- dynamic model of a combined cycle power plant.

This paper is an introduction to ThermoSysPro library, and presents the combined cycle power plant test-case.

Using dynamic models for combined cycle power plants (as well as for any other type of power plants) allows to go beyond the study of fixed set points to:

- check precisely the performances and the design given by the manufacturers (commissioning),
- verify and validate by simulation the scenario of large transients such as gas turbine trips,
- find optimised operating points,
- find optimised operation procedures,
- perform local and remote plant monitoring,
- build correction curves,
- etc.

In order to challenge the dynamic simulation capabilities of the library, a step load variation from 100% to 50% and a turbine trip (sudden stopping of the gas turbine) were simulated.

2 Introduction to ThermoSysPro

2.1 Objectives of the library

From the end-user's viewpoint, the objectives of the library are:

- Ability to model and simulate thermodynamic processes.
- Ability to cover the whole lifecycle of power plants, from basic design to plant operation and maintenance. This implies the ability to model detailed subsystems of the plant, and to model the whole thermodynamic cycle of the plant, including the I&C system.
- Ability to initialize the models for a given operating point. This is essentially an inverse problem: how to find the physical state of the system given the values of the observable outputs of the system.
- Ability to perform static calculations (for plant monitoring and plant performance assessment) and dynamic calculations (for operation assistance) faster than real time.
- Ability to fit the plant models against real plant data using for instance data assimilation techniques.
- Ability to use the models to improve the quality of measurements using the data reconciliation technique.
- Ability to use the models for uncertainty studies by propagating uncertainties from the inputs to the outputs of the model.

From the model developer's viewpoint:

- The library should be easy to read, understand, extend, modify and validate.
- The library should be sharable at the EDF level, and more.

- The library should be truly tool independent.
- The library should be stable across language and tools versions.
- The library should be validated against significant real applications.
- The library should be fully documented. In particular, all modeling choices should be clearly justified.

2.2 General principles of the library

The library features multi-domain modeling such as thermal-hydraulics (water/steam, flue-gases and some refrigerants), neutronics, combustion, solar radiation, instrumentation and control.

The library is founded on first physical principles: mass, energy, and momentum conservation equations, up-to-date pressure losses and heat exchange correlations, and validated fluid properties functions. The correlations account for the non-linear behaviour of the phenomena of interest. They cover all water/steam phases and all flue gas compositions. Some components such as the multifunctional heater contains correlations that were obtained from experimental results or CFD codes developed by EDF. An early Modelica implementation of the IAPWS-IF97 standard by H. Tummescheit is used for the computation of the properties of water and steam.

The level of modelling detail may be freely chosen. Default correlations are given corresponding to the most frequent use-cases, but they can be freely modified by the user. This includes the choice of the pressure drop or heat transfer correlations. Special attention is given to the handling of two-phase flow, as two-phase flow is a common phenomenon in power plants. The physics of two-phase flow is complex because of the mass and energy transfer between the two phases and the different flow regimes (bubbles, churn or stratified flow...) [4]. Currently, mixed and two-fluids 3, 4 and 5 equations flow models are supported. For instance, 3 equations are used for the homogeneous single-phase flow pipe model, 4 equations for the drum model, and 5 equations for the separated flow pipe model. The different flow regimes are accounted for by appropriate pressure drop and heat transfer correlations. The drift-flux model may be used to compute the phase velocities. Also, accurate sets of geometrical data are provided for some heat exchangers.

Flow reversal is supported in the approximation of convective flow only (the so-called upwind scheme where the Peclet number is supposed to be infinite [5]). It is planned to investigate the interest of taking diffusion into account for a more robust computation of flow reversal near zero-flow.

The components are separated into 4 groups: yellow components for static modelling only, green components for static and dynamic modelling, blue components for dynamic modelling only, and purple components for fast dynamic modelling (waterhammer). All components are compatible with each other, but many yellow components do not withstand zero-flows, so they cannot be used to model transients that involve flow reversal for instance. The green components group is composed of singular pressure losses in the approximation of zero-volume, so that the coefficients of the derivative terms of the balance equations are equal to zero. Hence, one should only use yellow or green components for static modelling only.

The library components are written in such a way that there are no hidden or unphysical equations, that components are independent from each other and to ensure as much as possible upward and downward compatibility across tools and library versions. This is particularly important in order to control the impact of component, library or tool modifications on the existing models.

To that end, only the strictly needed constructs of the Modelica language are used. In particular, the inheritance and stream mechanisms are not used, and no physical meaning is assigned to the fluid connectors: they are considered as a means to pass information between components, so they are not part of the physical equations.

The components are connected together using the fluid connectors according to the staggered grid scheme [5]. This scheme divides the components into two groups: volumes and flow models. Volumes compute the mass and energy balance equations, whereas flow models compute the momentum balance equations. Volumes may have any number of connectors, whereas flow models have exactly two connectors (they look like pipes, although they are not necessarily pipes). The staggered grid scheme states that flow models should be connected to volumes only, and volumes should be connected to flow models only. It is however possible to connect flow models together without breaking the staggered grid rule, by considering that the intermediate volume has a zero-volume capacity.

2.3 Structure of the fluid connectors

The structure of the fluid connectors is of particular importance as it reflects the overall structure of the library.

As already stated, the fluid connectors do not bear any physical meaning. They are only considered as a way to pass information between components, and

should therefore be eliminated from the physical equations system after compilation of the model. However, as connectors are sensitive to the components graph orientation rules, they define the convention for the sign of the flows, or in other words, which direction in the graph is assigned for positive flows, and which direction is assigned for negative flows.

For flow orientation, two alternatives are possible. The first is to have a flow orientation at the component level: the flow is positive when it enters the component, and negative when it leaves the component. The second is to have a flow orientation at the graph level: the flow is positive under normal operating conditions, and negative in case of reverse flow conditions, the latter being most often transitory. The second alternative has been preferred over the first one for the fluid connector, as it gives a flow sign convention closer to the end-user perception of the operation of the system.

From these requirements, and also from the fact that the staggered grid and the upward schemes are used, the structure of the connector follows.

```
connector FluidInlet
  SIunits.Pressure P;
  SIunits.SpecificEnthalpy h;
  SIunits.MassFlowRate m_flow;
  SIunits.SpecificEnthalpy h_flow;

  input Boolean a=true;
  output Boolean b;
end FluidInlet;
```

```
connector FluidOutlet
  SIunits.Pressure P;
  SIunits.SpecificEnthalpy h;
  SIunits.MassFlowRate m_flow;
  SIunits.SpecificEnthalpy h_flow;

  output Boolean a;
  input Boolean b;
end FluidOutlet;
```

There are actually an inlet connector and an outlet connector. These two connectors have the same physical structure (P, h, m_flow, h_flow), but different flow orientations enforced by the Booleans a and b. The flow is positive when entering the component at the inlet or leaving the component at the outlet. The keywords input and output prevent connecting inlets or outlets together (see Figure 1 where inlets are blue and outlets are red).

Notice that the Modelica prefix flow is not used for m_flow and h_flow. The reason is that (1) the mass balance and energy balance equations are not gener-

ated by the connections, but fully implemented in the volumes, (2) and that the sign convention for positive flows is not compatible with the sign convention of the Modelica flow prefix, which stipulates that all flows should be of the same sign (positive or negative) when entering the component via the connector. As a consequence, multiple connections are not allowed as in Modelica.Fluid for instance, so that mergers or splitters must be modeled using volumes. In practice, this is not considered as a restriction, as in most cases, mergers or splitters do have non trivial physical behaviours which could not be simply represented by multiple connections.

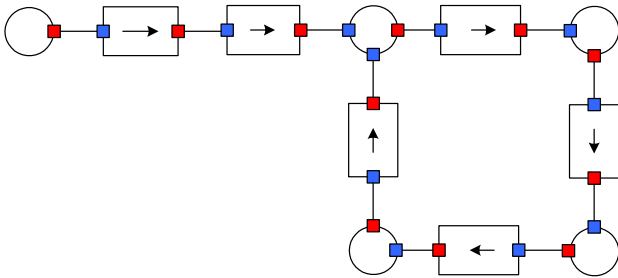


Figure 1: connecting components

P and h denote resp. the average fluid pressure and specific enthalpy inside the control volumes. m_flow and h_flow denote resp. the mass flow rate and specific enthalpy crossing the boundary between two control volumes (see Figure 2).

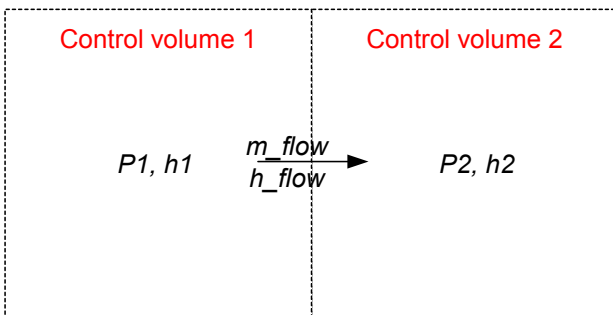


Figure 2: finite volume discretization

P, h and m_flow are the state variables of resp. the mass, energy and balance equations. h_flow is not a state variable. The purpose of h_flow is to compute the fluid specific enthalpy using the upwind scheme. P and h are computed within volumes, whereas m_flow and h_flow are computed within flow models (see Figure 3).

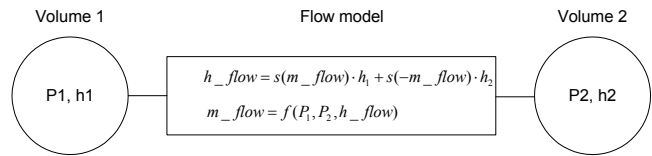


Figure 3: staggered grid scheme

According to the upwind scheme:

$$h_flow = s(m_flow) \cdot h_1 + s(-m_flow) \cdot h_2$$

where s denotes the step function:

$$s(x) = 0 \text{ if } x \leq 0 \text{ and } s(x) = 1 \text{ if } x > 0$$

2.4 Organization of the library

The library is subdivided into application domains. Each application domain corresponds to a connector type. Each application domain is divided into packages corresponding to broad component types: boundary conditions, connectors, heat exchangers, machines, pressure losses, sensors, volumes, etc. (see Figure 5 in the Appendix).

Components may be written in plain Modelica text, or constructed by connecting other components from the library, as shown in Figure 4.

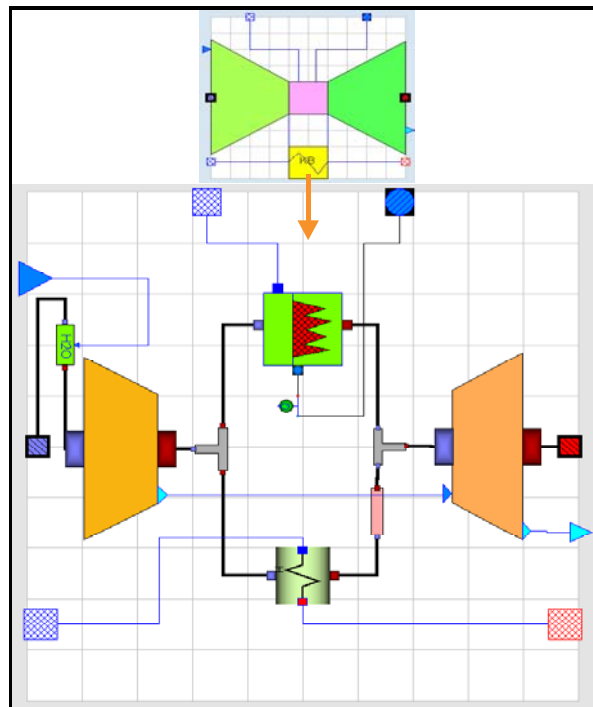


Figure 4: model component of a gas turbine

3 Model of the combined cycle power plant

3.1 Description of the model

Actually, two models are used: one to simulate the power generator step reduction load (see Figure 6 in the Appendix), the other to simulate the full GT trip (see Figure 7 in the Appendix). In the model used to simulate the GT trip, the gas turbine is replaced by a boundary condition.

The model contains two main parts: the water/steam cycle and the flue gases subsystem. Only one train is modelled, so identical behaviour is assumed for each HRSG and for each gas turbine.

HRSG model

The model consists of 16 heat exchangers (3 evaporators, 6 economizers, 7 super-heaters), 3 evaporating loops (low, intermediate and high pressure), 3 drums, 3 steam turbine stages (HP, IP and LP), 3 pumps, 9 valves, several pressure drops, several mixers, several collectors, 1 condenser, 1 generator, several sensors, sources, sinks and the control system limited to the drums level control.

An important feature of this model is that the thermodynamic cycle is completely closed through the condenser. This is something difficult to achieve, because of the difficulty of finding the numerical balance of large closed loops.

The list of component used for the development of the HRSG model is given in Table 1.

Table 1: library components used in the HRSG model

Type	Model name in the library
Condenser	DynamicCondenser
Drum	DynamicDrum
Generator	generator
Heat exchanger	DynamicExchangerWaterSteamFlueGases = DynamicTwoPhaseFlowPipe ExchangerFlueGasesMetal HeatExchangerWall
Pipe	LumpedStraightPipe
Pump	StaticCentrifugalPump
Sensor	SensorQ
Steam turbine	StodolaTurbine
Valve	ControlValve
Water mixer	VolumeB, VolumeC

Type	Model name in the library
Water splitter	VolumeA, VolumeD

Heat Exchanger : Flue Gases/ Water Steam

Based on first principles mass, momentum and energy balance equations, the following phenomena are represented:

- transverse heat transfer,
- mass accumulation,
- thermal inertia,
- gravity,
- pressure drop within local flow rate.

Drum and Condenser

Based on first principles mass and energy balance equations for water and steam, the following phenomena are represented:

- drum level and swell and shrink phenomenon,
- heat exchange between the steam/water and the wall,
- heat exchange between the outside wall and the external medium.

Steam turbine

Based on an ellipse law and an isentropic efficiency.

Pump

Based on the characteristics curves.

Pressure drop in pipes

Proportional to the dynamic pressure \pm the static pressure.

Mixer/splitter

Based on the mass and energy balances for the fluid.

GT model

The model consists of 1 compressor, 1 gas turbine, 1 combustion chamber, sources, sinks and 1 air humidity model.

The list of component models used for the development of the GT model is given in Table 2.

Table 2: library components used in the GT model

Type	Model name in the library
Air humidity	AirHumidity
Compressor	GTCompressor
Gas turbine	CombustionTurbine
Combustion chamber	GTCombustionChamber

Gas turbine

Based on correlations for the characteristic.

Compressor

Based on correlations for the characteristic.

Combustion chamber

Based on first principles mass, momentum and energy balance equations. The pressure loss in the combustion chamber is taken into account.

3.2 Data implemented in the model

All geometrical data were provided to the model (pipes and exchangers lengths and diameters, heat transfer surfaces of exchangers, volumes...).

The plant characteristics are given below.

Gas Turbine (GT)

Compressor compression rate: 14

Steam Generator (HRSG)

HRSG with 3 levels of pressure.

High pressure circuit at nominal power: 128 bar

Intermediate pressure circuit at nominal power: 27 bar

Low pressure circuit at nominal power : 5.7 bar

Steam Turbine

High pressure at nominal power : 124.5 bar, 815 K

Intermediate pressure at nominal power : 25.5 bar, 801 K

Low pressure at nominal power : 4.8 bar, 430 K

Condenser

Steam flow rate: 194 kg/s

Water temperature at the inlet: 300 K

3.3 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 characteristics of the pumps,
- the ellipse law coefficients of the turbines,
- the isentropic efficiencies of the turbines,
- the friction pressure loss coefficients of the heat exchangers and of the pipeline between the equipments,
- the CV of the valves and the valves positions (openings).

3.4 Simulation scenarios

For simulation runs, two scenarios were selected.

The first scenario is a power generator step reduction from 100 to 50% load:

- Initial state (combined cycle): 100 % load
- Final state (combined cycle): 50% load (800 s slope)

The second scenario is a full GT trip (sudden stopping of the gas turbine):

- Initial state (GT exhaust): 894 K, 607 kg/s
- Final state (GT exhaust): 423 K, 50 kg/s (600 s slope)

The following phenomena are simulated:

- flow reversal,
- local boiling or condensation,
- swell and shrink effect in drums,
- drums levels and condenser level,
- drums pressure control

3.5 Simulation scenarios

Simulation runs were done using Dymola 6.1.

The simulation of the scenarios were mostly successful. However, some difficulties were encountered when simulating large transients, mainly stemming from the large size of the model:

- poor debugging facility,
- slow simulation,
- large number of values to be manually provided by the user for the iteration variables,
- no efficient handling of these values.

In particular, it has been observed that sometimes Dymola cannot calculate the initial states, even when all iterations variables are set very close to their solution values. This was the main difficulty that was encountered when closing the loop through the condenser.

When Dymola stops before the end of the simulation, no clear message is delivered to analyse the causes of the failure.

Tool improvements were analysed and reported as part of the EDF contribution to the EUROSYSLIB project, in partnership with Politecnico di Milano [6].

3.6 Simulation results

The model is able to compute precisely:

- the air excess,
- the distribution of water and steam mass flow rates,
- the thermal power of heat exchangers,

- the electrical power provided by the generator,
- the pressure temperature and specific enthalpy distribution across the network,
- the drums levels and the condenser level,
- the performance parameters of all the equipments,
- the global efficiencies of the water/steam cycle and gas turbine.

The computational time is faster than real time (with Dymola 6.1).

The results of the simulation for 100% load are given below.

Gas Turbine (GT)

Nominal power: 2*230 MW,

Steam Generator (HRSG)

Thermal power: 2*350 MW,

Steam Turbine

Nominal power: 275 MW,

Condenser

Thermal power: 423 MW.

Outlet water temperature: 306 K

Vacuum pressure: 6100 Pa

The results of the simulation runs are given in Figure 8 and Figure 9 in the Appendix. They are consistent with the engineer’s expertise.

However, when the GT trip reaches full stopping of the plant, the recirculation flows in the evaporators do not go to zero as expected, for reasons that are not yet fully understood.

4 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. Among other test-cases, a dynamic and rather large model of a combined cycle power plant has been developed to validate the library. This model comprises the flue gas side and the full thermodynamic water/steam cycle closed through the condenser. Two difficult transients were simulated: a step reduction load of the power generator and a full gas turbine trip. The results are mostly consistent with the engineer’s expertise.

Despite of some simulation difficulties because of the lack of debugging tools for Modelica models, this work shows that the library is complete and robust enough for the modelling and simulation of complex power plants. However these two essential qualities for a power plant library should continuously be improved and maintained in the long run.

Acknowledgements

This work was partially supported by the pan-European ITEA2 program and the French government through the EUROSYSLIB project.

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] Souyri A., Bouskela D., ‘Pressurized Water Reactor Modelling with Modelica’, Modelica 2006 conference proceedings.
- [4] Collier J.G., and Thome J.R., ‘Convective Boiling and Condensation’, Mc Graw-Hill Book Company (UK) limited, 1972 Clarendon Press, Oxford, 1996.
- [5] Patankar S.V., ‘Numerical Heat Transfer and Fluid Flow’, Hemisphere Publishing Corporation, Taylor & Francis, 1980.
- [6] Casella F., Bouskela D., ‘Efficient method for power plant modelling’, EDF report H-P1C-2010-01929-EN, 2010.
- [7] David F., Souyri A., Marchais G., ‘Modelling Steam Generators for Sodium Fast Reactors with Modelica’, Modelica 2009 conference proceedings
- [8] El Hefni B., Péchiné B., ‘Model driven optimization of biomass CHP plant design’, Mathmod conference 2009, Vienna, Austria.

Appendix

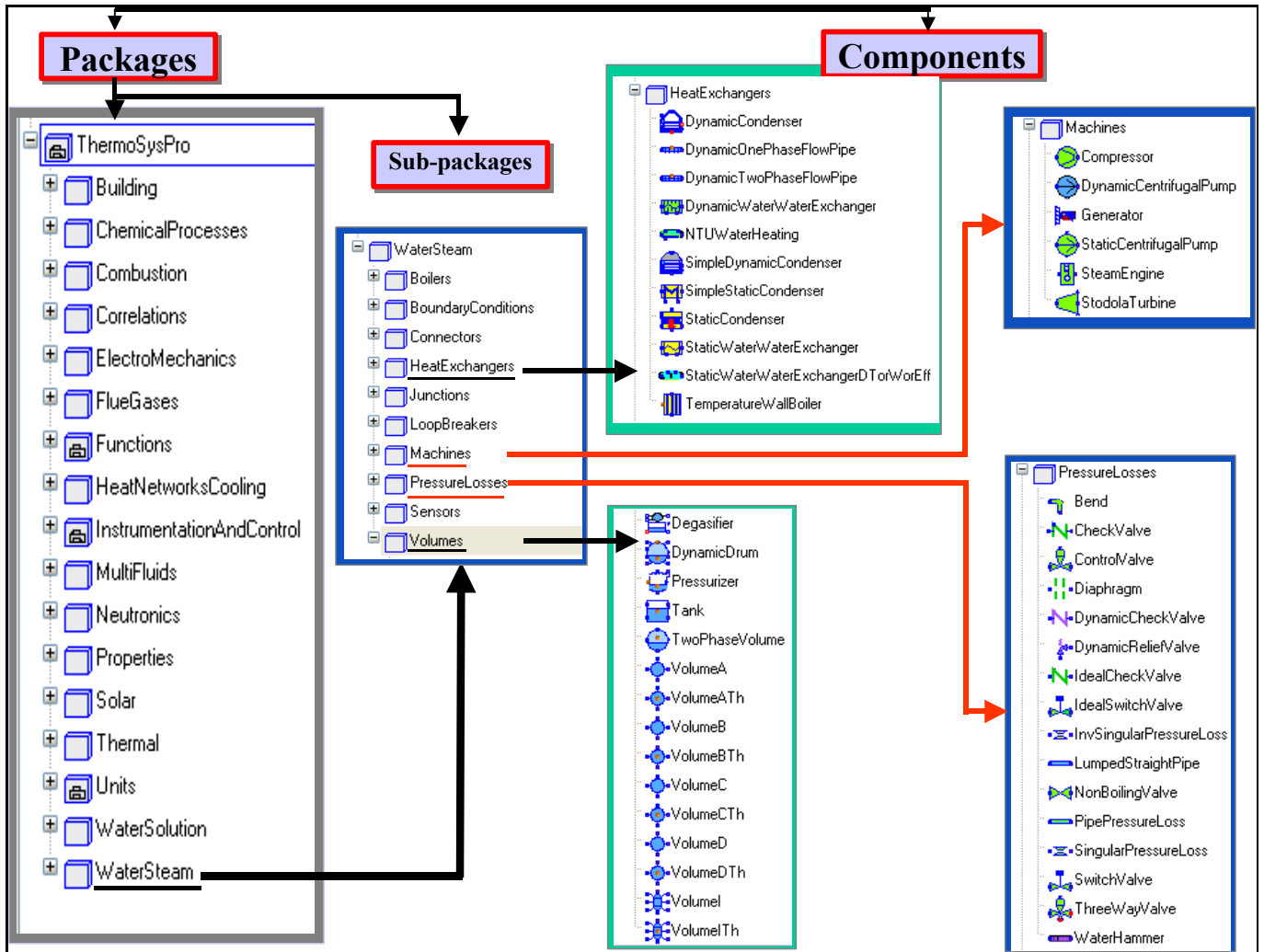


Figure 5: organization of the library

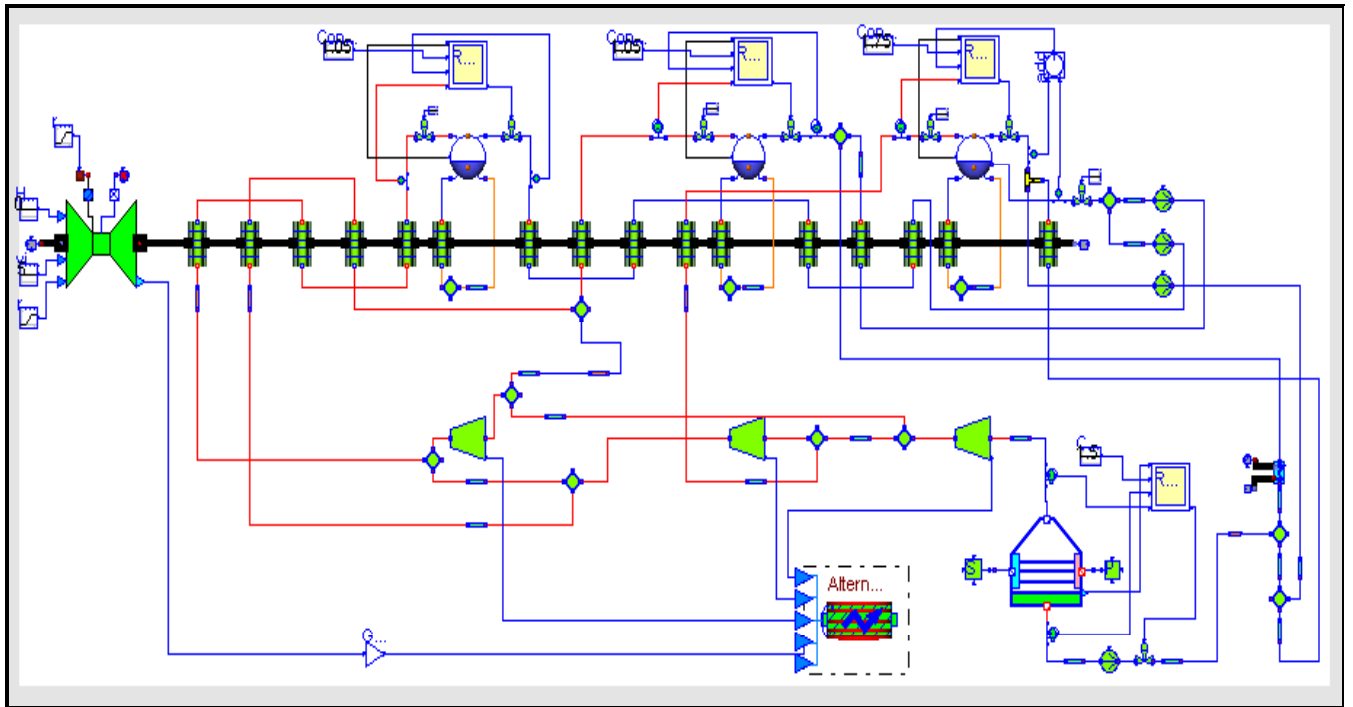


Figure 6: model of the combined cycle power plant used for the power generator step reduction load

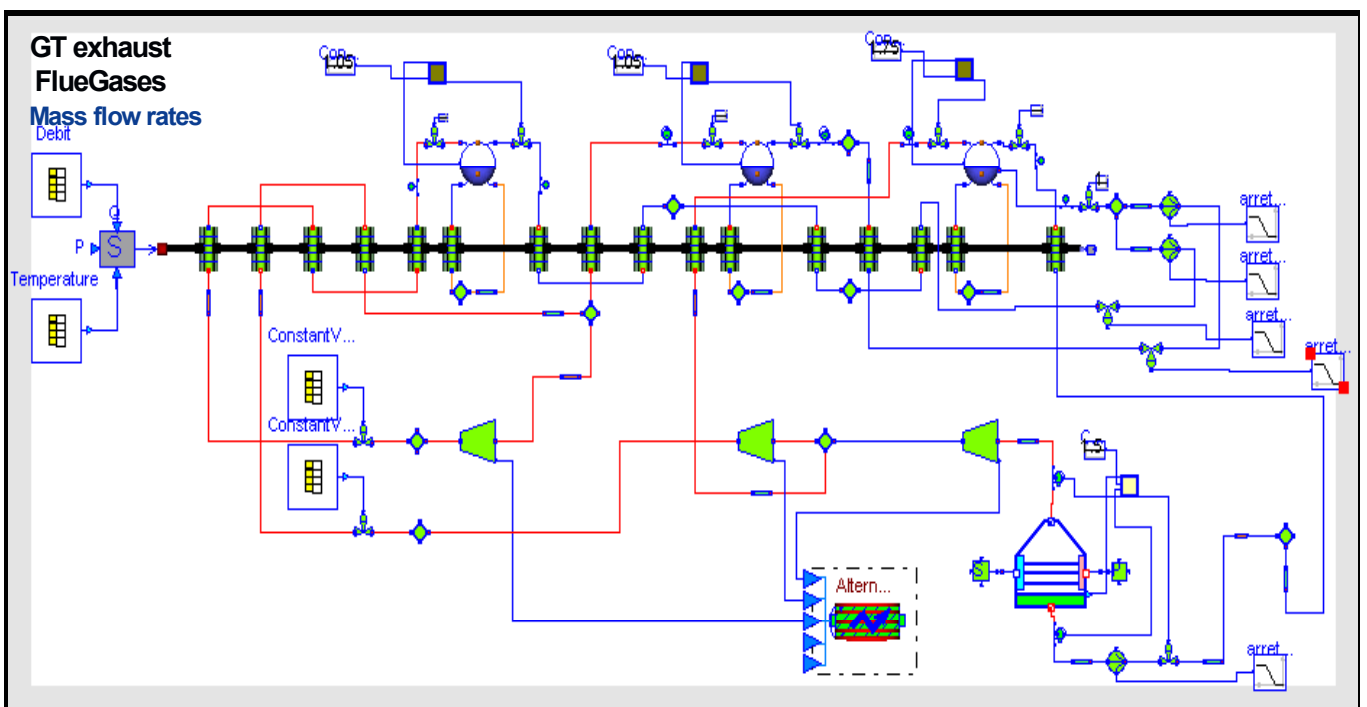


Figure 7: model of the combined cycle power plant used for the full GT trip

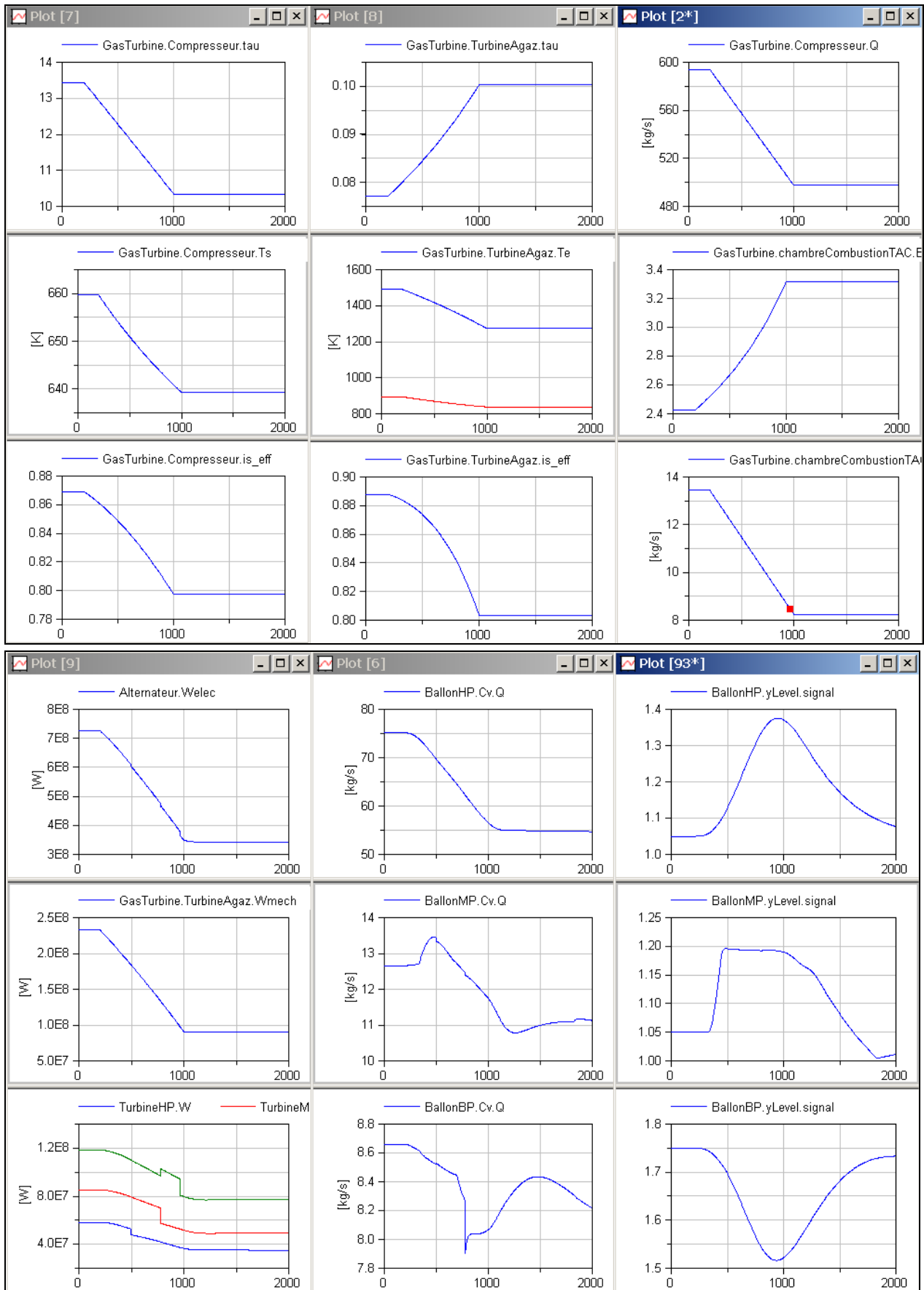


Figure 8: power generator step reduction simulation (-50%)

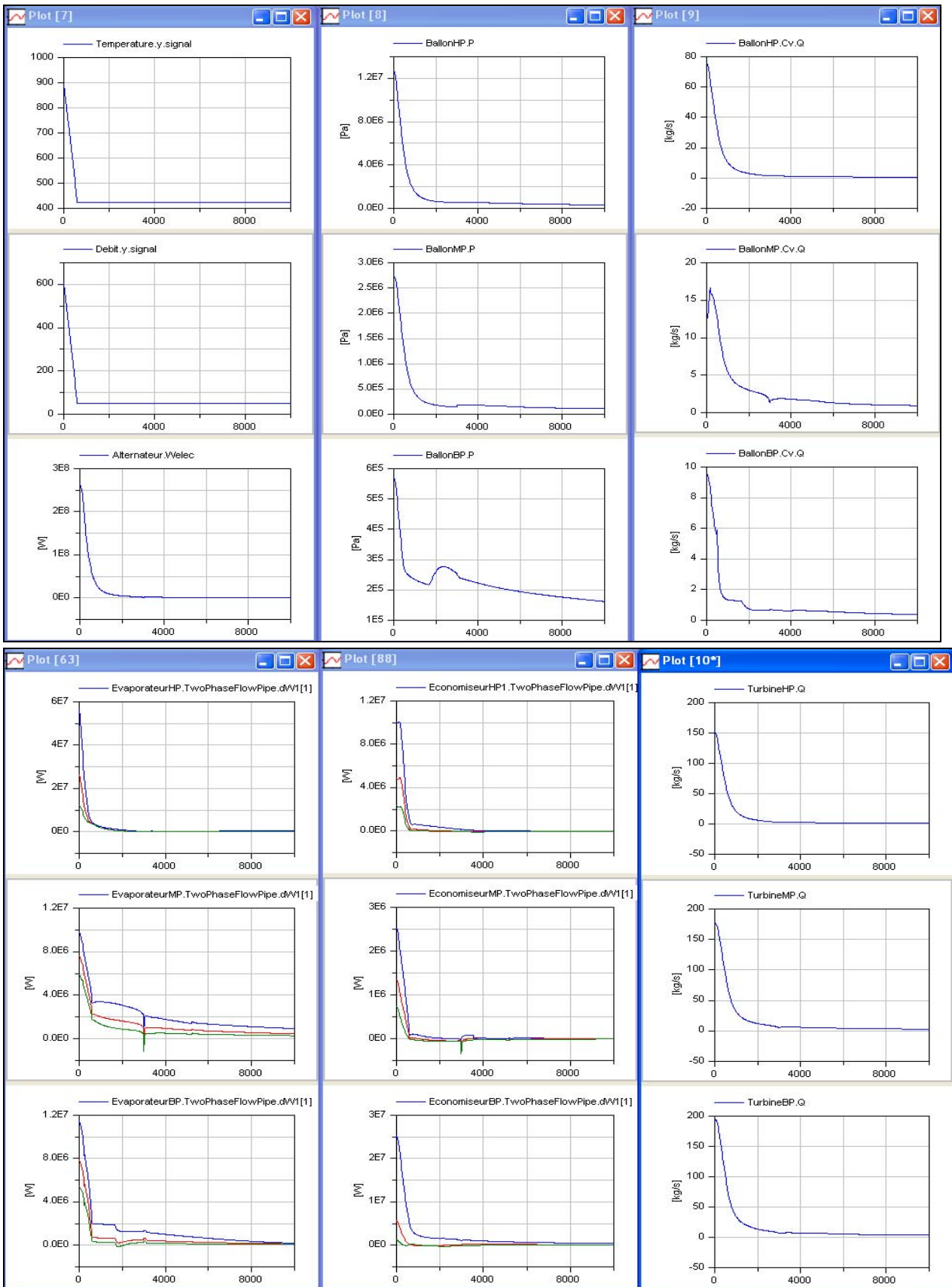


Figure 9: GT trip simulation

Implementation of Models for reheating processes in industrial furnaces

Daniel Rene Kreuzer Andreas Werner

Vienna University of Technology, Institute of Energy systems and Thermodynamics
Getreidemarkt 9/302, 1060 Wien

Abstract

Components were developed for the modeling of industrial furnaces in the iron and steel industry like pusher-type and walking beam furnaces. A cell model on the basis of the dynamic pipe model of the Modelica Fluid Library was designed. The cell model computes the heat transfer between furnace walls, flue gas and the processed steel goods. The radiative heat transfer is modeled by a 1-dimensional method based on Hottel's net radiation method. Furthermore, models for furnace walls, slabs, hearth and the transport of the slabs were designed. The models are suitable for analyzing operation modes and designing control concepts

Keywords: reheating; furnace; simulation; radiation

1 Introduction

Currently the manufacturing industry has focused on increasing energy efficiency of their production processes. The effective and sustainable use of energy is becoming more important, due to the issues of climate change and therefore the demand of reducing CO_2 emissions. The iron and steel industry in particular incorporates a multiplicity of reheating processes in the production chain of their commodities. For instance the reheating of slabs in pusher type or walking beam furnaces to set up the right temperature interval before hot rolling as well as annealing of steel coils in continuous or batch furnaces to trigger the micro structure and therefore the properties of the steel products. The furnaces mentioned are commonly operated by using natural gas and off gas from blast furnace, coke oven and basic oxygen furnace. While in the past the development of reheating processes was concentrated on increasing the production output, recently the decrease in energy consumption became more and more important. Raising energy efficiency requires the optimization or redesign of the reheating process steps,

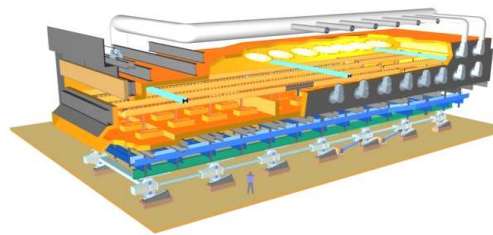


Figure 1: Walking beam furnace

e.g. by implementing advanced control concepts or by analyzing and assessing different operation modes. These engineering tasks indicate the need for accurate dynamic furnace models. There exists a variety of furnace models as e.g. described in [5],[8], [4], [2]. Several models which are used by the industry are based on experimental modeling (system identification) or so called black box modeling. The disadvantage of such models is the restricted validity to the considered systems. The main intention of the presented work is the design of highly reusable models contained within a library which are able to simulate the unsteady heat transfer phenomena of reheating steel goods in industrial furnaces. Due to the focus on modeling the physical phenomena and the reuse ability of the models the Modelica language standard and the simulation environment Dymola were chosen for the modeling task.

2 Basic considerations

Figure 1 shows the layout of a typical reheating furnace which can be used for heating semi-finished steel products before hot rolling. Basically they work as counter current heat exchangers, therefore the transport direction of the processed goods is di-

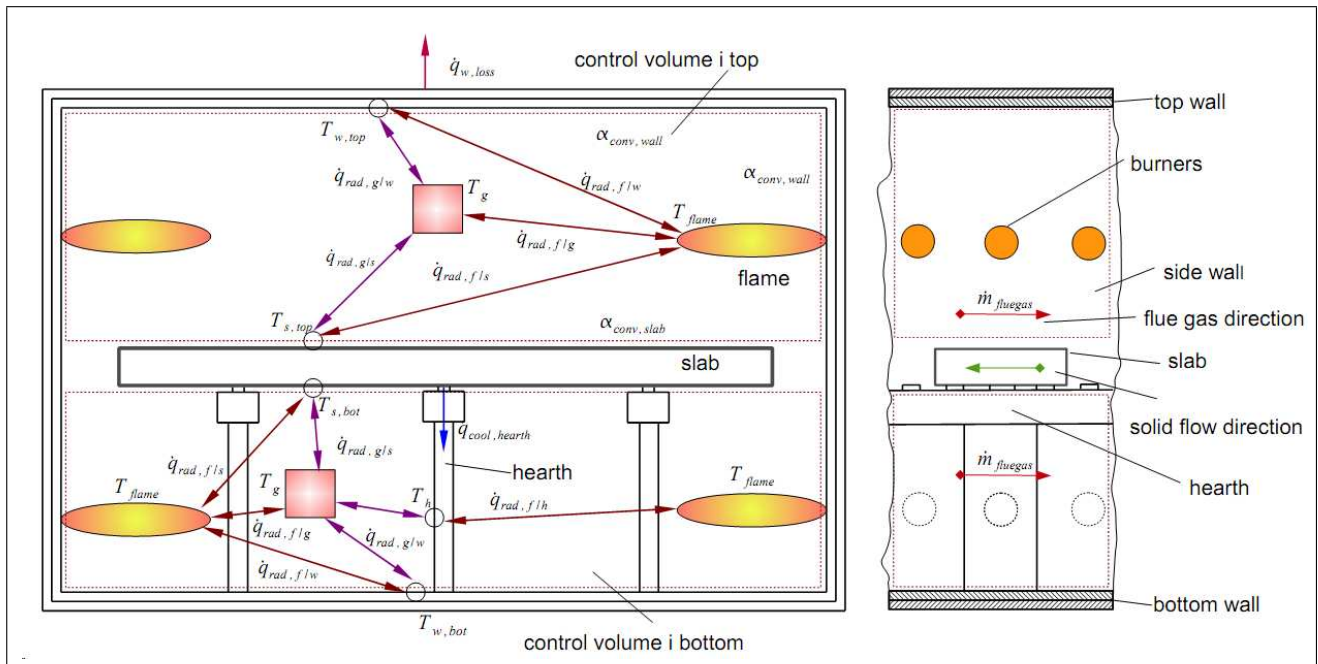


Figure 2: Heat transfer mechanisms of reheating process

directed against the main flow of the flue gas. Reheating furnaces usually consist of a convective, heating and soaking zone. The heating and soaking zone are commonly operated by gaseous fuel-type burners, and the main heat transfer phenomenon is radiation. Due to the high flue gas temperatures up to 1400 °C, furnaces are lined with refractory. The walls have a typical multiple layer configuration of different materials. During processing, the steel goods are positioned on a hearth, which has to be cooled for e.g. by evaporative cooling. The basic phenomena of reheating processes which have to be modeled are

- Heat transfer due to radiation between walls, flue gas, flame and processed goods
- Heat transfer due to convection between flue gas and walls as well as flue gas and processed goods
- Heat conduction in furnace walls, processed goods and hearth
- Combustion of gases (burners)
- Transport of flue gas - Fluid Flow Models (FFM)
- Transport of processed goods - Solid Flow Models (SFM)

Figure 2 depicts the appearing heat flow rates of reheating slabs in a walking beam furnace. The further presented models consider primarily the reheating of

slabs in walking beam or pusher-type furnaces. However the models for heat transfer phenomena are generally valid and may only need a minor adaptation for use at other reheating processes. The major diversities in modeling reheating processes are evident for the SFM and their impact on the heat transfer models.

3 Fluid Flow

The general concept of the reheating process models is based on a homogeneous cell model which incorporates fluid flow and heat transfer. The dynamic pipe flow model of the Modelica.Fluid.Library is used for modeling the 1-dimensional fluid flow of flue gases in the furnaces. A comprehensive description of solving the fluid transport equations with the Modelica.Fluid.Library is given in [3]. Properties of the fluids are determined by the Modelica.Media.Library, e.g. internal energy, specific enthalpy, density, thermal conductivity, dynamic viscosity. The heat transfer between the gases, the walls and the slabs is realized by designing a new heat transfer model which replaces the standard models.

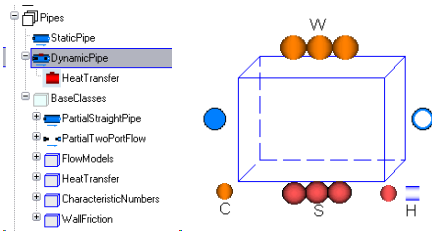


Figure 3: Cell model based on Modelica fluid library

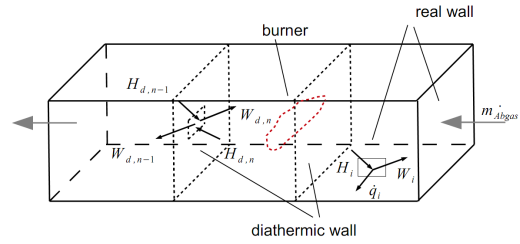


Figure 5: Discretization of a combustion chamber

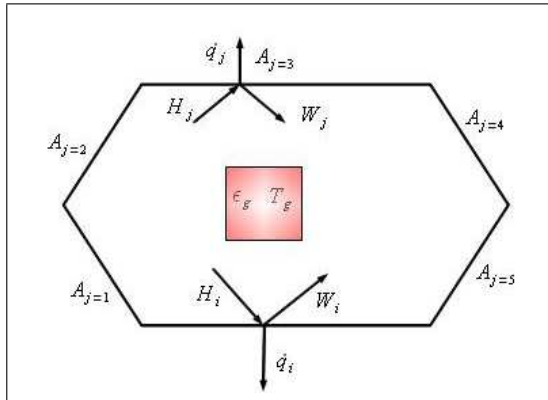


Figure 4: Net radiation heat flow of surface i in a cell containing the emitting gas

4 Heat Transfer

4.1 Radiative heat transfer

The radiative heat transfer plays a key role in modeling high temperature industrial furnaces. The different mathematical radiation models can be classified by their dimensionality. In literature 0-dimensional, 1-dimensional and 3-dimensional models are described. 3D-models achieve the highest accuracy but require high efforts in modeling and computing time, those kind of models are usually implemented in commercial CFD-packages. 0D-models are based on the stirred-tank reactor and give only a rough estimation for the radiative heat transfer of the regarded systems. 1D-models are in between the two aforementioned types and seem to be a proper trade-off in accuracy and modeling effort. For modeling the radiative heat transfer a 1D-model derived by [6] was selected. It has been developed from the net radiation method of Hottel, which is described in [7], [1]. Scholand [6] depicts the equations needed for calculating radiative heat exchange between radiating surfaces in an enclosure comprising isothermal radiative gases.

4.1.1 Governing Equations

The following section states the equations according [6] used for the radiative heat transfer model. The grey and diffuse radiating surface A_i , shown in Figure 4, receives the radiative heat flow H_i composed of the heat flow emitted by the different grey surfaces and the gas. The grey surface i again emits with its temperature $\epsilon_i E_{bi}$ and reflects $H_i \rho_i$. The net-radiation heat flow of the surface i , which exchanges heat with n different surfaces of the enclosure, yields

$$\dot{q}_i = W_i - H_i \quad (1)$$

at which the outgoing radiative heat flow is given by

$$W_i = \epsilon_i \cdot E_{bi} + \rho_i \cdot H_i \quad (2)$$

ϵ_i is the emissivity of the grey wall and $E_{bi} = \sigma \cdot T_i^4$ is the hemispherical total emissive power of a black body whereas σ is the Stefan-Boltzmann constant and T_i the surface temperature. The incoming radiative heat flow is given by

$$H_i = \epsilon_{gi} \cdot \sigma T_g^4 + \frac{1}{A_i} \sum_{j=1}^n A_j W_j F_{ji} \tau_{ji} \quad (3)$$

and equation 1 results in

$$\dot{q}_i = \epsilon_i \cdot E_{bi} - \epsilon_i \cdot \epsilon_{gi} \cdot \sigma T_g^4 - \epsilon_i \cdot \frac{1}{A_i} \sum_{j=1}^n A_j W_j F_{ji} \tau_{ji} \quad (4)$$

ϵ_{gi} is the emissivity of the gas and T_g is the gas temperature. W_j is the outgoing heat flow of the j^{th} -surface element, A_i and A_j are the associated surface areas, F_{ji} is the view configuration factor between surface j and surface i and τ_{ji} is the transmittance of the gas. The discretization of a furnace is done by connecting several of the above described cells. The heat transfer due to radiation between those cells is realized by a diathermic wall approach depicted in figure 5. The diathermic wall presents the interface between

two cells, e.g. the relation between cell n and the cell $n - 1$ yields:

$$H_{i,n} = W_{i,n-1} \quad (5)$$

$$H_{i,n-1} = W_{i,n} \quad (6)$$

The above stated equations presume a grey gas entirely surrounded by grey walls. For a grey gas the dependency of emissivity, absorptivity and transmittance is given by $\alpha_g + \tau_g = 1$, $\alpha_g = \varepsilon_g$ and $\rho_g = 0$, whereas for a grey body $\tau = 0$, $\alpha + \rho = 1$ and $\varepsilon = \alpha$. A real gas like flue gas from a combustion reaction is composed of several species which have different properties. Important radiative gases like CO_2 and H_2O are selective emitters which means their emissivity, absorptivity and transmittance strongly depend on the wave length interval. Due to this fact the assumption of a single grey gas surrounded by grey walls has to be abandoned whereas the assumption of grey body radiation of the walls remains valid. Therefore the equation 4 yields:

$$\dot{q}_i = \varepsilon_i \cdot E_{bi} - \varepsilon_i \cdot \int_{\lambda=0}^{\infty} \varepsilon_{\lambda gi} \cdot \sigma T_g^4 d\lambda \quad (7)$$

$$- \varepsilon_i \cdot \frac{1}{A_i} \int_{\lambda=0}^{\infty} \sum_{j=1}^n A_j W_j F_{ji} \tau_{ji} d\lambda \quad (8)$$

The clear-gray gas model approach according to [9] is used for the calculation of the emissivity, absorptivity and transmittance of the gas mixture. The emitting species are considered as grey gases and the other species are concentrated as non emitting clear gas. The emissivity of a gas mixture with H_2O and CO_2 as emitting gases is hence:

$$\varepsilon_G = \sum_{i=1}^3 a_i (1 - e^{-k_{gi} p_g S_G}) \quad (9)$$

where k_{gi} is the absorption coefficient of the different constituents, p_g equals the sum of the partial pressure of p_{H_2O} and p_{CO_2} , S_G is the radiation beam length and a_i are weighting factors which are given by a linear approach

$$a_i = b_{0i} + b_{1i} \cdot T_G \quad (10)$$

and the conditional equation

$$\sum_{i=1}^3 a_i = 1 \quad (11)$$

The absorptivity is calculated with a similar approach to equation 9 and 10, only the weighting factors are determined with the wall temperatures instead of the gas temperature. For solving the equations of the radiative heat transfer the calculated radiative gas properties are applied to equation 4.

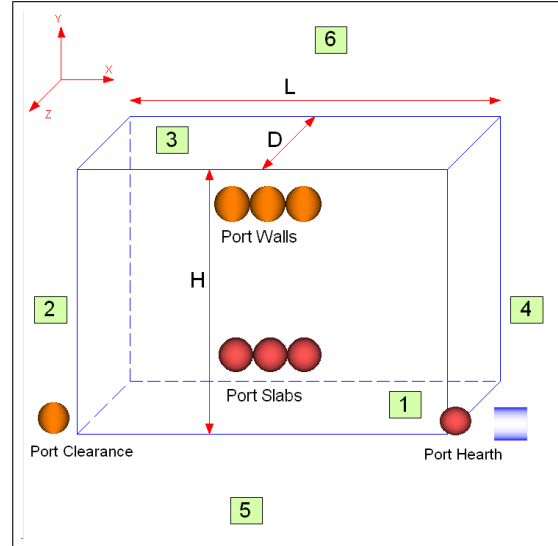


Figure 6: Cell model for heat transfer

4.2 Convective heat transfer

The convective heat transfer can simply be added to the net-radiation equation of the surface i

$$\dot{q}_i = W_i - H_i + \dot{q}_{conv} \quad (12)$$

$$\dot{q}_{conv} = \alpha \cdot (T_g - T_i) \quad (13)$$

The heat transfer coefficients for the single surfaces are determined with the relation of forced convection at a single plate according to [9]

4.3 Thermal conduction

The transient heat conduction problem in walls, slabs and hearth is modeled by Fourier's 1-dimensional partial differential equation

$$\frac{\partial T}{\partial t} = a \cdot \frac{\partial^2 T}{\partial x^2} + \frac{\dot{Q}_e}{\rho c_p} \quad (14)$$

4.4 Implementation of heat transfer models

4.4.1 Radiation and convection models

The net-radiation method and the convective heat transfer equations presented in section 4.1 and 4.2 were integrated into a new designed model `RadiationAndConvectionHeatTransfer`. The model calculates all the data necessary for the heat transfer mechanisms, e.g. the view configuration factors of the involved bodies, the radiative properties of the gases, the Nußelt-Numbers, the convective heat transfer coefficients etc. Those data in conjunction with the temperatures lead finally to the net-radiation

```

for k in 1:n loop
  Q_dot_radconv[k] = sum(q[i] * A_agg[i] for i in 1:m+7) + sum(q_res[j] * A_agg[j] for j in 1:m+7)/n;
end for;

```

Figure 9: Code for computation of $Q_{b_dot_radconv}$ for a single cell

```

connector HeatPort_Radiation
  "Port for heat transfer due to radiation for real surfaces and diathermic surfaces"
  Modelica.SIunits.Temperature T "Port temperature";
  Modelica.SIunits.Temperature T_g "Port temperature";
  flow Modelica.SIunits.HeatFlowRate Q_flow
  "Heat flow rate (positive if flowing from outside into the component)";
  flow Modelica.SIunits.HeatFlowRate Q_flow_res
  "Heat flow rate (positive if flowing from outside into the component)";
  a;
end HeatPort_Radiation;

```

Figure 7: HeatPort_Radiation Connector

```

connector GeometryPort_Radiation
  Real D "propagation in z-direction";
  Real H "propagation in y-direction";
  Real L "propagation in x-direction";
  Real origin[3] "origin of radiant surfaces";
  Real epsilon "emissivity of surfaces";
  Integer d[3] "type of surface";
  Integer attitude
  "attitude of surface; 1 perpendicular to x, 2 perpendicular to y, 3 perpendicular to z";
end GeometryPort_Radiation;

```

Figure 8: GeometryPort_Radiation Connector

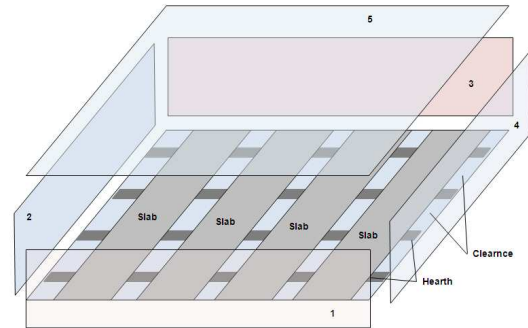


Figure 10: Surface configuration of top cell

and convective heat flows. A bottom-cell and a top-cell are developed for the discretization of the furnaces to take the different geometric configurations into account. For the energy exchange between fluid, walls, slabs, hearth and the neighbouring cells a new type of connectors HeatPort_Radiation was developed. It is not working like real physical connector due to the fact that it may be connected to diathermic walls. A diathermic wall usually represents the interface between two discretized cells. Unlike a real wall a diathermic wall has to enable a heat flow in both directions at the same time. Therefore another flow variable Q_flow_res and the corresponding potential variable T_g which is equal to the gas temperature of the cell were introduced. The radiative heat transfer depends on the temperature differences of the involved bodies and it depends on the position, attitude, propagation and emissivity of the emitting surfaces. That fact leads to a further none physical connector GeometryPort_Radiation which transmits the necessary data from the involved bodies to the heat transfer model and vice versa. The generated connectors are combined to composite connectors. Figure 6 shows the heat transfer model of a top-cell and the connections to walls, slabs and the hearth. Two further connectors are present one which sends a signal with needed data for heat conduction between slabs and hearth and the other one which realizes a heat flow due to radiation between a top and a bottom cell via a diathermic wall. The connection between the heat transfer model and the fluid flow model is made via the energy balance

for a cell stated by [3] for a single flow segment i .

$$\text{der}(Us[i]) = Hb_flows[i] + Ib_flows[n] + Qb_flows[i]$$

Figure 9 shows the code for the computation of vector $Q_dot_radconv$ for n different flow segments. An accurate implementation of heat transfer allows only the discretization of one flow segment. The vector Qb_flows is set equal to the vector $Q_dot_radconv$ which is determined from the heat flow vectors q and q_res of the participating $m+7$ surfaces. The number of surfaces is determined from the number of slabs m which are present during the simulation, the number of furnace walls, the hearth surface and the surface of the gap between slabs and hearth. For all real walls q_res is equal to 0 and q is the net heat flow due to radiation and convection. The net heat flow q of the slabs is set to 0 if the slab is not present in the considered cell. The vector A_agg aggregates the surface areas of all participating bodies.

4.4.2 Wall, hearth and slab models

The wall hearth and slab models are based on the Modelica.Thermal.HeatTransfer.Components, which provide a solution for the 1-dimensional heat conduction equation. Figure 11 shows the model of a layered wall which is discretized with the elements HeatCapacitor and ThermalConductor. The models have connectors to the heat transfer model which represents the connection to the radiative and convective trans-

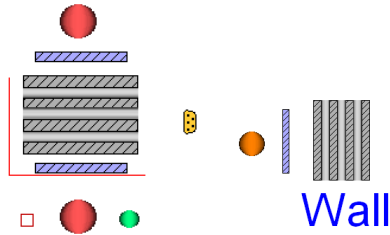


Figure 11: Layered slab and wall model

fer equations and a connector `HeatPort` of the `Modelica.Thermal.HeatTransfer.Components` which serves to set up the boundary conditions. The interface between the connectors of the cell and the `Modelica.Thermal.HeatTransfer.Components` is realized by a special surface element. This element represents the emitting surface and is the first discretization of the wall and it is based on the equations of a the `HeatCapacitor`. The slabs and the hearth are modeled in the same way as the layered wall. The slab model possesses two surface models for the heat transfer between a top cell and a bottom cell, further it has connections to the hearth for the heat transfer due to conduction and radiation as well as to the solid flow model which determines the slab position. The hearth model is split into a model for a top cell and a bottom cell and presents an equivalent system for the real configuration, because the exact geometry configuration can hardly be modeled due to the characteristics of a 1-dimensional analysis. The `HearthBottom` model has a connector for the cell and one for a boundary condition which is usually a temperature condition. The `HearthTop` model possesses connectors for the heat conduction with the slabs, the radiative and convective heat transfer with the cell, the radiative heat transfer with the slabs and a signal connector which transmits needed data for computing the phenomena mentioned before. Figure 13 shows the parameter needed for a layered wall. The number of layers, the number of discretization per layer, the thickness of the single layers and the material properties of each layer as well as the start temperatures of the different layers have to be supplied.

5 Combustion

The release of energy in reheating furnaces is typically realized by the combustion of gaseous fuels. The fundamental combustion calculation is utilized to deter-

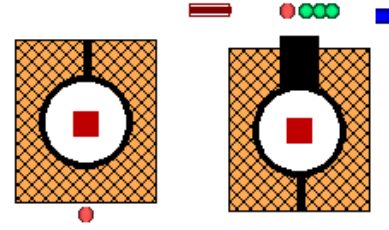


Figure 12: Hearth model for bottom and top cell

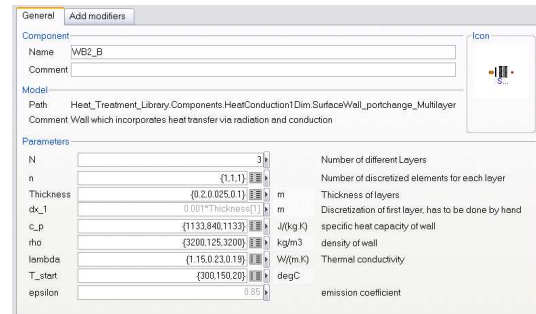


Figure 13: Parameter window for layered wall

mine the required mass flow of air, the flue gas amount and the flue gas composition. The energy input of the fuels is considered by the lower heating value H_u . The designed models neglect the dynamics of the combustion reactions. The `PartialLumpedVolume` serves as base class for the combustion model. The energy balance for the combustion volume is given by

$$\frac{\partial(\rho u V)}{\partial t} = \dot{m}_{Air} \cdot h_{Air} + \dot{m}_{Fuel} (h_{Fuel} + H_u) - \dot{m}_{FlueGas} h_{FlueGas} + \sum \dot{Q} \quad (15)$$

For the adiabatic combustion $\sum \dot{Q} = 0$. For instance, $\sum \dot{Q}$ can be used to define heat transfer due to flame radiation but the geometric configuration and the propagation of the flame has to be known as well as the flame emissivity. The volume model has connections for the combustion air, the gaseous fuel and the flue gas. It is assumed that the entire volume is filled with homogeneously distributed flue gas. The flue gas in the volume model is completely combusted without any residues of combustibles. Three new ideal gas mixtures were defined with the `Modelica.Media.Library`. The mixture `FuelGaseous` which consists of the species H_2 , N_2 , CO_2 , CO , H_2S , CH_4 , C_2H_2 , C_3H_8 , C_4H_{10} , C_3H_6 , C_2H_2 and C_4H_8 , `FlueGas` which consists of O_2 , Ar , N_2 , CO_2 , H_2O , SO_2 and `Air` which consists of O_2 , Ar , N_2 , CO_2 , H_2O . The combustion is realized by computing the rate of change dependent on time of the single flue

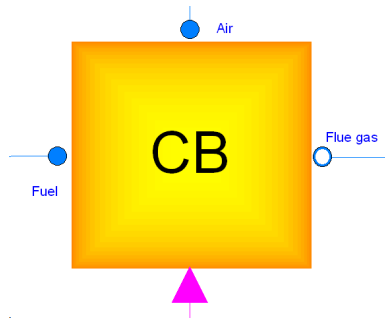


Figure 14: Volume for combustion of gases

gas masses $\frac{\partial m_{O_2}}{\partial t}$, $\frac{\partial m_{N_2}}{\partial t}$, $\frac{\partial m_{Ar}}{\partial t}$, $\frac{\partial m_{CO_2}}{\partial t}$, $\frac{\partial m_{H_2O}}{\partial t}$, $\frac{\partial m_{SO_2}}{\partial t}$ according to the fundamental combustion equations and a predefined excess air.

6 Solid Flow Model

The solid flow model consists of 3 different models shown in figure 15. One model represents the heating goods, in this case slabs and therefore the SlabDistributor. It possesses the same connectors as the slabs and aggregates all slabs, which are present during simulation. The number of slabs m has to be specified before starting a simulation and determines how many objects "slab" are present. Every single object slab has to be connected to each heat transfer cell, which is done via the SlabDistributor. The SlabFeedmodel is the most important element for the solid flow because it computes the position of the single slab models, refers the slab properties to the slab models and triggers the slab feed movement. Computing the slab positions and the heat transfer between slabs and the other involved bodies led to the introduction of a reference coordinate system. Therefore all models which represent geometric objects, e.g. the cells, walls slabs etc., get an origin referenced to the introduced coordinate system. The propagation of the models is specified into the positive direction of the $x(1)$ -, $y(2)$ - and $z(3)$ -axis. The fluid flow is defined positive in the positive x -direction and the solid flow is directed against the positive x -axis. The SlabFeed has a connection to the SlabDistributor and transmits slab position and properties. A further connection is needed via boolean input to a pulse generator. At every pulse the slabs are moved one step forward and a new slab is fed into the furnace or cell. The slab transportation occurs step wise which means they are allocated to a certain discrete position at each time step of the simulation. That means that the time of slab movement is equal to 0. Due to the nature of a pusher

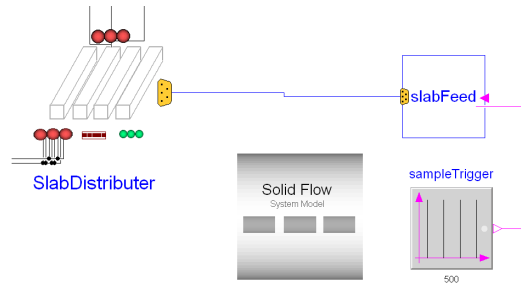


Figure 15: Solid Flow Models

type or a walking beam furnace all slabs are moved simultaneously through the cells as long as they are in the furnace. If they leave the boundaries of the furnace the slabs are brought to an end position. If more than m slabs are processed during one simulation the position of the completed slabs is reset to a starting position, new property values and new input temperatures are set. The SolidFlowSystem model is defined as an inner system wide component which provides important data for most of the models, e.g. maximum number of slabs in simulation, slab properties, general furnace data etc..

7 Testing the heat transfer models

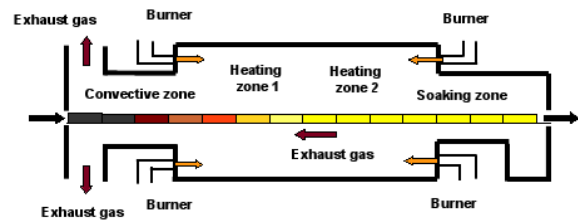


Figure 16: Configuration of reheating furnace

The heat transfer models are tested by modeling three zones of a reheating furnace depicted in figure 16. The regarded furnace incorporates two heating zones, a convective zone and the soaking zone. The reheating furnace is operated by natural gas and the energy input is evenly distributed over the two heating zones. The combustion air is preheated in a recuperator, which is not considered in the model.

7.1 Experimental setup

For the experimental setup the two heating zones and the convective zone are modeled. The soaking zone is neglected because it serves only to ensure that the

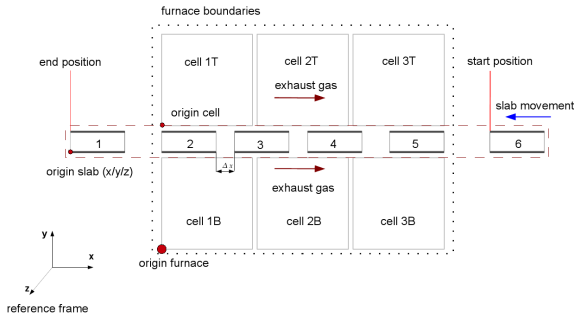


Figure 17: Discretization of furnace

core of the slab reaches the target temperature. Figure 17 shows the discretization of the furnace zones. Each zone is discretized by two cell models, a top cell model and a bottom cell model. The heating zone 1 is represented by cell2T and cell2B, heating zone 2 by cell1T and cell1B and the convective zone by cell3T and cell3B. The postfix 'B' and 'T' distinguish between bottom and top cells. The heating zones are equipped with 12 burners, which are modeled by the burner elements. One burner model represents 6 burners and is connected to either a top or a bottom cell. The cells of the convective zone (cell3T and cell3B) are not connected to burner elements. The top cells and bottom cells are connected via the fluid connectors and the diathermic walls among each other. Top and bottom cells are not connected via fluid connectors, there exists no fluid exchange between the top and the bottom cells. There is only an exchange of heat due to radiation between the top and the bottom cells. Furthermore the cells are connected to the wall models, the slab models and the hearth models. Figure 18 and 19 show the burner elements and the connected cell model. Figure 20 shows the whole configuration in Dymola.

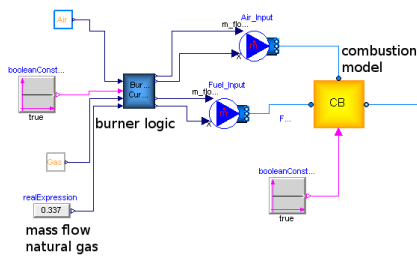


Figure 18: Model of burner elements

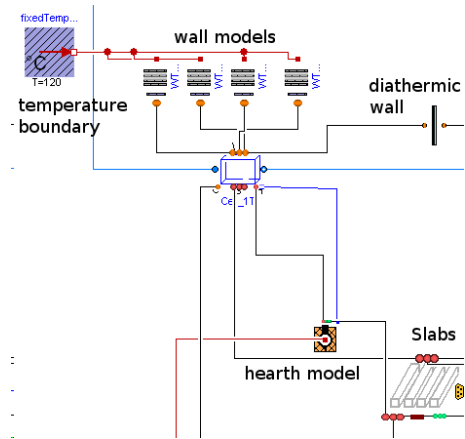


Figure 19: Model of connected cell and wall models

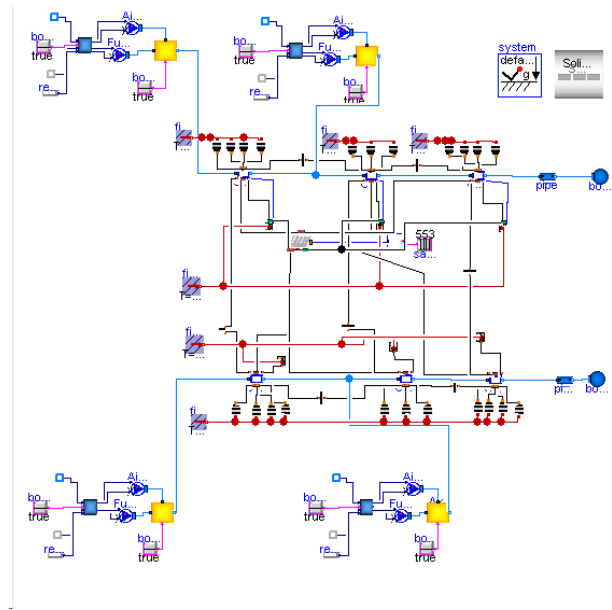


Figure 20: Model of the furnace zones in Dymola

7.2 Simulation

For the simulation of the modeled furnace configuration the number of slab objects was set to $m=25$. That implies a maximum number of 25 slabs can be simultaneously processed by the furnace. Each slab is discretized into 5 layers subdividing the thickness of the slab and two surface elements. It is necessary to set start positions and temperatures for the slabs, which is basically done via the solid flow model. The walls are built of different layers of refractory, which are common for those type of furnace, each layer is equal to a discretization of the heat conduction model. The outside connectors of the walls and further the wall models need proper temperature start values. The cells have a fixed position according to a reference coordinate system and values for the propagation in x-, y-

and z-direction. The position of the cell origin is important for the calculation of the view factors. The furnace has a length of 26.5 m in x-direction. The heat conduction between slabs and the hearth had to be prevented because of high values of the cell dimensions (8m to 10.5m) in x-direction compared to the width of the slabs ($\approx 1.5m$). The heat conduction between the slabs and the hearth of each cell is computed with the mean temperature of all slabs which are present in a cell. The energy input via the burners is set constant for the whole simulation time at 68.75MW, the heating value of the natural gas is set to 48.9kJ/kg, the single constituents of the gas remain constant during the simulation and the temperature of the preheated combustion air is set to 350°C. The emissivity of all walls and the hearth is set to 0.9 and of the slabs to 0.8. The slab movement starts after 500s and after the duration of 553s the slabs are moved one step forward as well as a new slab is fed into the furnace. The value of the forward step is determined by the x-dimension of the incoming slab. 15000s of the furnace operation are simulated by using the integration algorithm RadauIIa for stiff systems. The integration time for the simulation of 15000s lasted 1711.39s.

7.3 Results

Figure 21 shows the temperature distribution over the simulation time of two different slabs. Figure 22 shows the discrete position of the slab origin over simulation time of the slab 19, furthermore it depicts the position during a whole pass of the slab element through the furnace. The dimensions of slab 19 at that time are $L(x) = 1.397m$, $H(y) = 0.218m$ and $D(z) = 12.12m$. At 1053s the slab 19 is fed into the furnace. The processing of the slab ends at 11560s which gives a time of residence of about 10507s. At the end of processing slab 19 has a surface temperature of 1351°C at the top and a temperature of 1335°C in the middle layer. The surface temperature at the bottom shows to be similar compared to the top temperature. The temperature rise of the slabs show also the transport from one zone into another by a change in the temperature slope. After the heating process slab 19 is fed to an end position where new properties and dimensions can be defined, e.g. a new input temperature. Afterwards the slab is fed to a predefined position in front of the furnace where it remains until processing is started again. Figure 24 shows the temperature distribution of the gas and the top wall of the top cells. It can be seen that the temperature is strongly influenced by the incoming and outgoing slabs. During the slab movement a

high amount of energy contained in the outgoing slab is leaving the system and another much lower amount of energy connected to the incoming slab is entering the system. The resulting difference in enthalpy leads to a temperature drop at each slab movement. Figure 25 shows the emissivity and the absorptivity of gas for the irradiating top wall of cell1T. The emissivity depends on the gas temperature but it is independent from the wall temperature. As expected it decreases with increasing temperature. The absorptivity of the gas depends on the temperature of the irradiating wall, as a result it has not the same values as the emissivity of the gas.

8 Conclusion

Components for modeling dynamic heat transfer in industrial furnaces like pusher-type or walking beam-type have been developed. They can be used to model single zones or entire furnace systems. Those physical models would be particularly suitable for designing advanced control concepts for the operation of the furnaces. The simulation of the test configuration shows satisfactory results. The most important future task is the validation of the heat transfer models by modeling an existing reheating furnace and comparing the simulation with measurement results.

9 Symbols

Symbol	Physical value
$A_i(m^2)$	surface area of wall i
$a_i(-)$	weighting factor gas radiation
$b_{0i}(-)$	coefficient of weighting factor polynomial
$b_{1i}(1/K)$	coefficient of weighting factor polynomial
$c_p(J/kgK)$	specific heat capacity
$E_b(W/m^2)$	hemispherical total emissive power of a black body
$F_{ij}(-)$	view configuration factor from surface i to j
$h(J/kg)$	specific enthalpy
$H_i(W/m^2)$	incoming radiative heat flow of surface i

Symbol	Physical value
$H_u(J/kg)$	lower heating value
$k_{gi}(1/m \cdot bar)$	absorption coefficient of species i
$\dot{m}(kg/s)$	mass flow
$\dot{Q}_e(W)$	power source
$\dot{q}_i(W/m^2)$	net-radiation heat flow of surface i
$\dot{q}_{conv}(W/m^2)$	convective heat flow of surface i
$S_G(m)$	radiation beam length
$T_i(K)$	temperature of wall i
$T_g(K)$	gas temperature
$u(J/kg)$	specific internal energy
$V(m^3)$	volume
$W_i(W/m^2)$	outgoing radiative heat flow of surface i
$\alpha_i(-)$	absorptivity of surface i
$\alpha_g(-)$	absorptivity of gas
$\varepsilon_g(-)$	emissivity of gas
$\varepsilon_i(-)$	emissivity of surface i
$\lambda(m)$	beam length
$\rho_i(-)$	reflectivity of surface i
$\sigma(W/m^2K^4)$	Stefan-Boltzmann constant
$\tau(-)$	transmittance of gas

[5] KIM, M. Y. A heat transfer model for the analysis of transient heating of the slab in a direct-fired walking beam type reheating furnace. *International Journal of Mass and Heat Transfer* 50 (2007), 3740–3748.

[6] SCHOLAND, E. *Ein einfaches mathematisches Modell zur Berechnung des Strahlungswärmeaustausches in Brennkammern*, vol. Fortschr.-Ber. VDI-Z Reihe 6 Nr.111. VDI-Verlag GmbH-Düsseldorf, 1982.

[7] SIEGEL, R., AND HOWELL, J. R. *Thermal Radiation Heat Transfer*. Hemisphere Publishing Corporation/Taylor & Francis, 1992.

[8] STRACKE, H. *Mathematisches Modell für brennstoffbeheizte Industrieöfen unter besonderer Berücksichtigung der Gasstrahlung*, vol. Forschungsberichte des Landes Nordrhein-Westfalen Nr. 2821/Fachgruppe Maschinenbau Verfahrenstechnik. Westdeutscher Verlag, 1979.

[9] VEREIN DEUTSCHER INGENIEURE, V.-G. V. U. C., Ed. *VDI-Wärmeatlas 10. Auflage*. Springer-Verlag, 2006.

References

[1] CLARK, J. A., KORYBALSKI, M. E., AND ARBOR, A. Algebraic methods for the calculation of radiation exchange in an enclosure. *Wärme und Stoffübertragung* 7 (1974), 31–44.

[2] DIETZ, U. *Einsatz mathematischer Modelle zur Simulation industrieller Feuerräume unter besonderer Berücksichtigung des Strahlungsaustausches*. PhD thesis, Universität Stuttgart, Institut für Verfahrenstechnik und Dampfkesselwesen, 1991.

[3] FRANKE, R., CASELLA, F., SIELEMANN, M., PROELSS, K., OTTER, M., AND WETTER, M. Standardization of thermo-fluid modeling in modelica.fluid. In *7th Modelica Conference, Como, Italy, Sep. 20-22, (2009)*.

[4] HACKL, F. *Eine betriebsorientierte Methode zur Berechnung der instationären tehermischen Vorgänge in Wärmeöfen*. PhD thesis, Montanistische Hochschule Leoben, 1974.

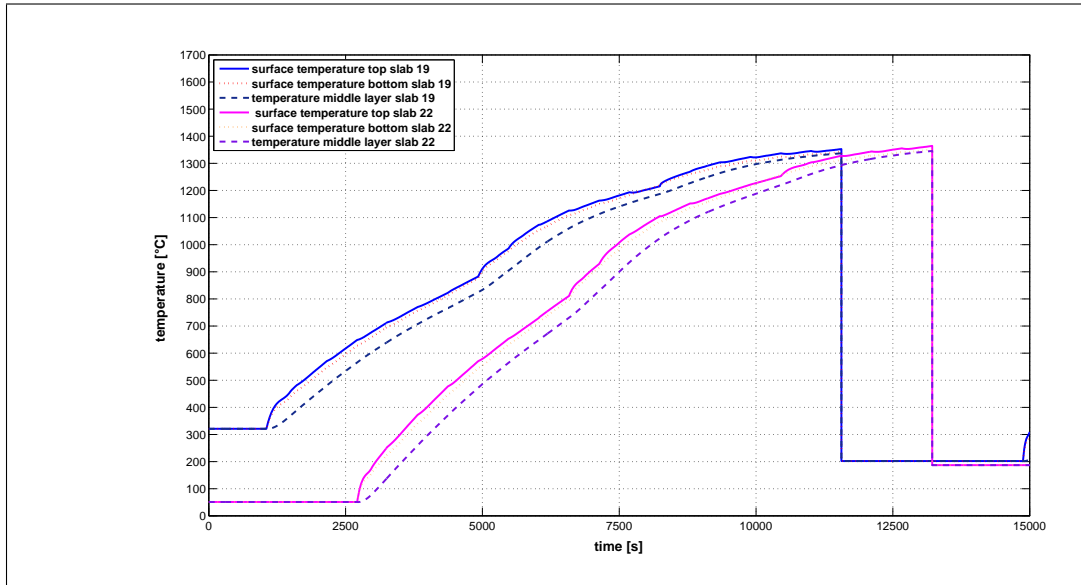


Figure 21: Slab temperature

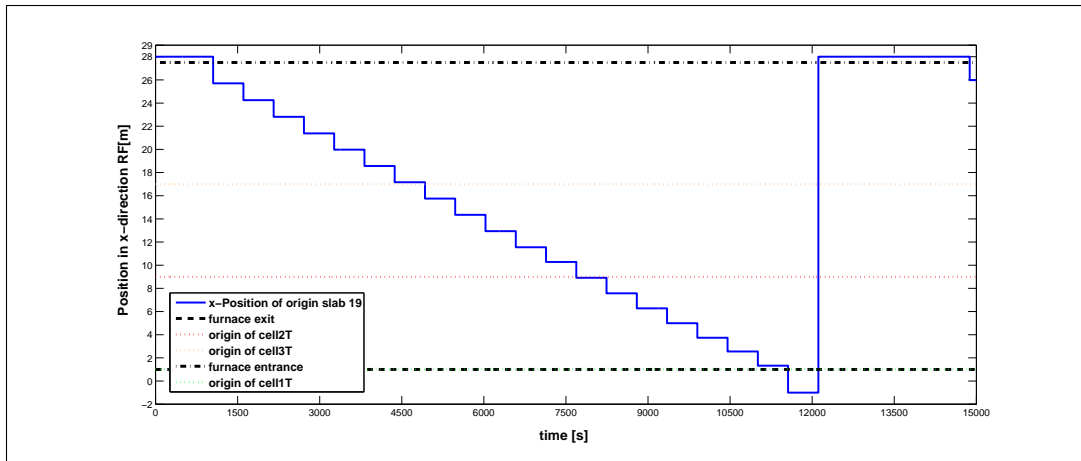


Figure 22: Discrete Slab positions during simulation

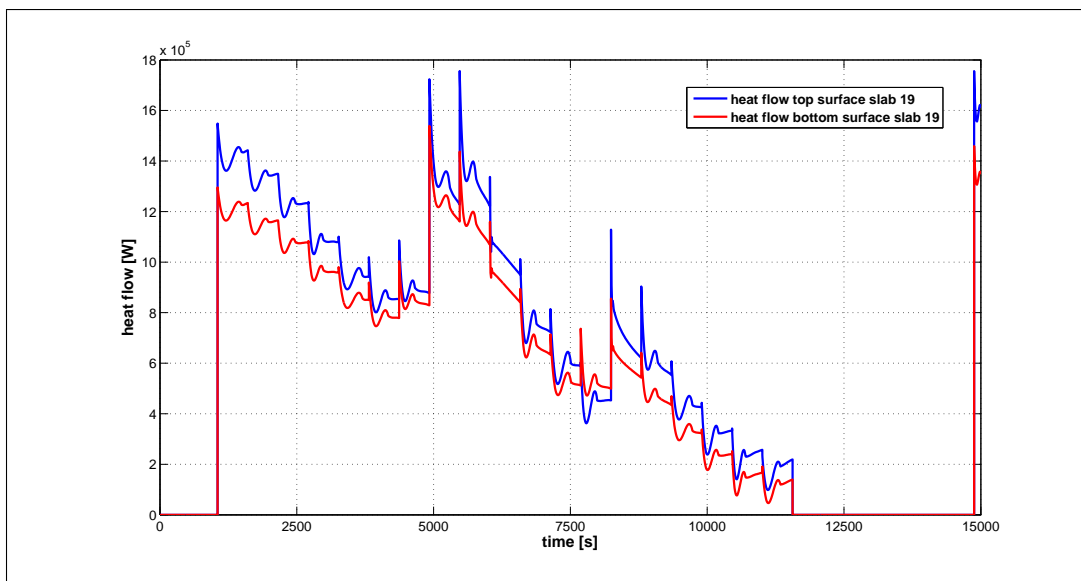


Figure 23: Heat flow at top and bottom surface of slab 19

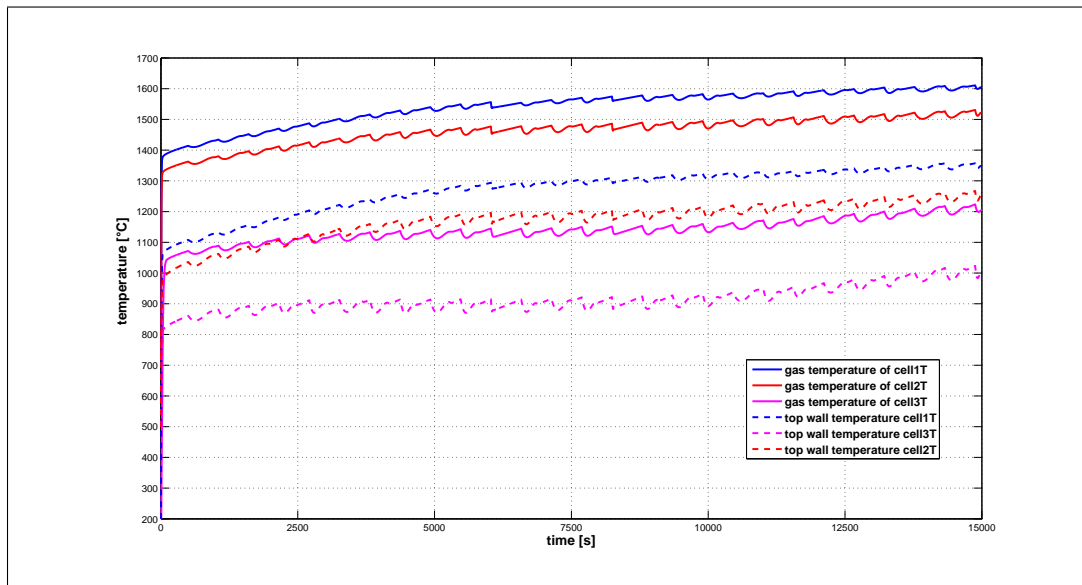


Figure 24: Gas and wall temperature of top cells

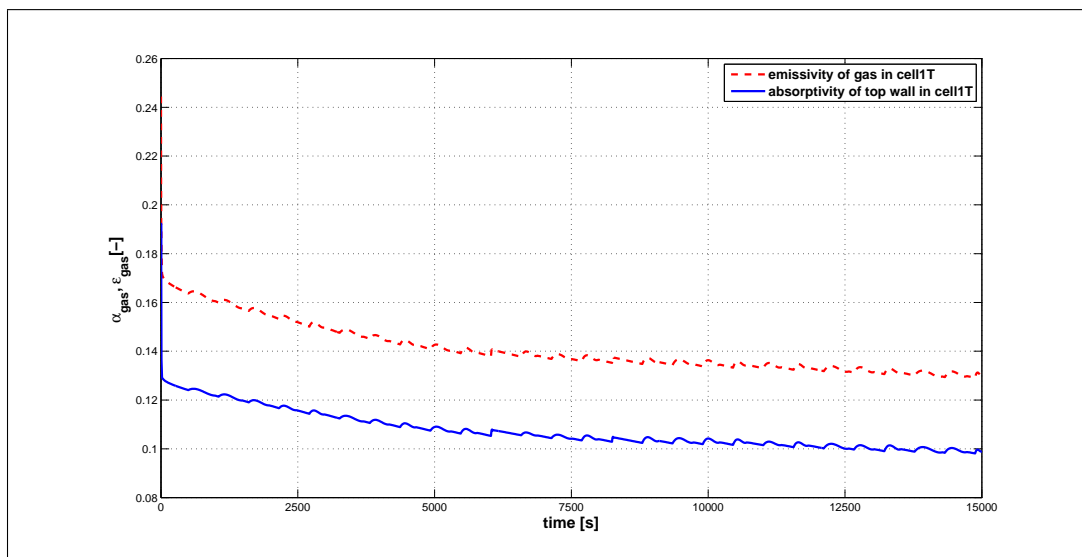


Figure 25: Gas emissivity and absorptivity

Modeling a Mains connected PWM Converter with Voltage-Oriented Control

Anton Haumer Christian Kral
 AIT Austrian Institute of Technology GmbH
 Giefinggasse 2, 1210 Vienna, Austria

Abstract

The majority of industrial controlled induction machine drives are connected to the mains via a diode bridge. However, if reduction of harmonic currents and / or regenerative operation is desired, replacing the diode bridge by an active front-end (AFE) is required. This paper describes two models of an AFE: a power balance model disregarding switching effects, and an ideal switching model of a pulse width modulation (PWM) converter. Both models are controlled utilizing space phasors in a voltage oriented reference frame. Voltage oriented control (VOC) of the mains converter can be compared with field oriented control (FOC) of a machine converter. Design and parametrization of the main parts—synchronization with mains voltage, current controller and DC voltage controller—are described in detail. Additionally, simulation results proving the implementation and demonstrating possible investigations as well as an outlook on further enhancements are presented.

Keywords: Active Front-End, PWM Converter, Voltage-Oriented Control

1 Introduction

The standard solution for industrial controlled drives consists of an induction machine, fed by a voltage source inverter. The drive usually is field-oriented controlled, as described e.g. in [1]. In many cases, the DC link is connected to the mains via a diode bridge due to cost saving reasons. However, some goals cannot be fulfilled by a diode bridge:

- Regenerative / recuperative operation
- Reduction of harmonic currents
- Control of reactive power

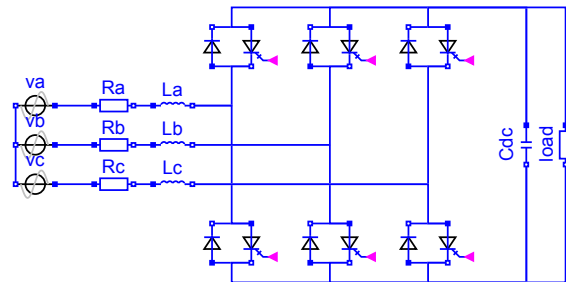


Figure 1: Model of an AFE with PWM converter

Replacing the diode bridge by an IGBT bridge as shown in Fig. 1 allows to address these topics. Between the mains and the AFE a mains reactor has to be used to limit the current slope and current harmonics, respectively.

The control of such an active front-end can be designed in several ways, as described in [2, 3, 4, 5, 6, 7]. In this paper, the voltage-oriented control with cascaded current control and DC voltage control has been chosen. This control scheme is comparable to the field-oriented control of the machine converter:

- Synchronization to: mains voltage position – field position
- Current controller for the space phasor
- DC voltage controller – speed controller

Both VOC and FOC utilize space phasor representation of voltages and currents; space phasor transformation is defined by

$$\underline{v} = \frac{2}{3} (v_1 + \underline{a}v_2 + \underline{a}^2v_3) \quad (1)$$

$$\underline{a} = e^{j\frac{2\pi}{3}} \quad (2)$$

$$v_0 = \frac{1}{3} (v_1 + v_2 + v_3) \quad (3)$$

with respect to a static reference frame. This transformation is power invariant:

Mains voltage line-to-line	400	V rms
Mains frequency	50	Hz
Short-circuit apparent power	35	MVA
Short-circuit power factor	0.2	-
Inductance of mains reactor	400	μH
Resistance of mains reactor	25	$\text{m}\Omega$
Nominal AC current	140	A rms
Switching frequency	5	kHz
DC capacitance	30	mF
Nominal DC voltage	693	V

Table 1: System parameters

$$p = \frac{3}{2} (+v_{Re}i_{Re} + v_{Im}i_{Im}) \quad (4)$$

$$q = \frac{3}{2} (-v_{Re}i_{Im} + v_{Im}i_{Re}) \quad (5)$$

Transformation of the space phasor to a rotating reference frame (index K) is done according to

$$\underline{v}_K = \underline{v}e^{-j\varphi} \quad (6)$$

$$\varphi = \varphi_0 + \int \omega_K dt \quad (7)$$

However, the product rule of differentiation has to be obeyed, which leads to:

$$\frac{d\underline{i}}{dt} = \frac{d\underline{i}_K}{dt}e^{+j\varphi} + j\omega_K\underline{i}_Ke^{+j\varphi} \quad (8)$$

The parameters used for all simulations are summarized in Tab. 1. To achieve more realistic simulations, the grid is not only modeled as a stiff voltage source, but an inductance and a resistance according to the grid's short-circuit apparent power have been added.

2 AFE Modeling

An AFE controls a power conversion bridge such way, that desired AC active and reactive currents are obtained. The following components have been implemented in a library, additionally summarized in ready-to-use models (Fig. 2):

- AC measurement box `mBoxAC` (subsection 2.1)
- mains reactor `mainsReactor`
- six pulse bridge `b6`, being either an instance of `PowerBalance` or `IdealSwitching` (subsection 2.2)

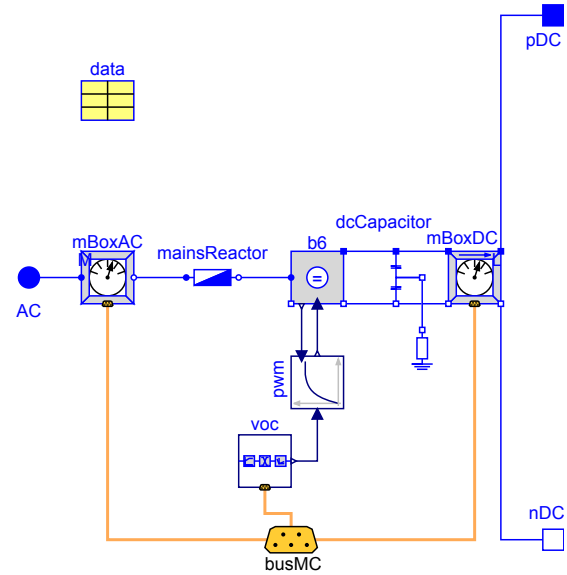


Figure 2: Mains converter with power balance model

- either a `pwmDelay` or a `spaceVectorPWM` block (subsection 2.2)
- buffer capacitor `dcCapacitor`; grounding has to be used carefully, to avoid undefined potentials or short-circuits; therefore, a grounding via a high earthing resistor is included
- DC measurement box `mBoxDC` (subsection 2.3)
- `voc` voltage oriented control block (subsection 2.4)
- `busMC` signal bus (subsection 2.5)
- parameter record `data` (subsection 2.6)

To the signal bus, either a reference input for active and reactive current is required, or a `vDCController` block (subsection 2.7) has to be connected. The DC controller takes the reference parameters or inputs for the set points of the DC voltage and the reactive AC current, controlling the AC active current to obtain the desired DC voltage.

2.1 MBoxAC

The `MBoxAC` measures AC voltages and currents. The voltage can either be measured at the mains terminals (i.e. mains reactor input), or at the converter AC input terminals (i.e. mains reactor output). For the latter case, the voltage drop across the mains reactor is added to the measured converter input voltage to acquire the desired mains voltage.

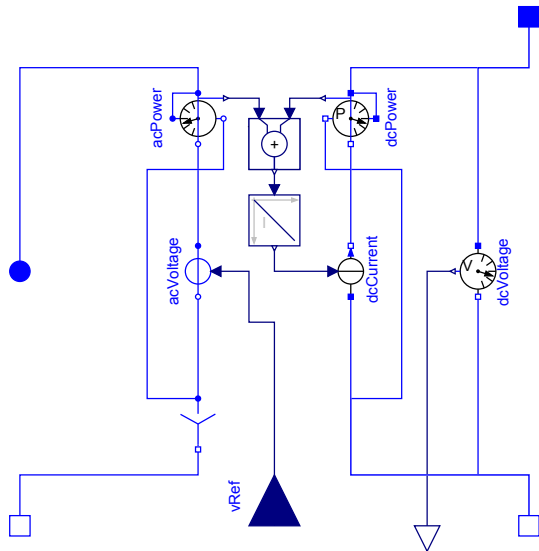


Figure 3: Power Balance model

According to subsection 3.1, reference angle and angular velocity of the mains voltage are determined by means of a synchronization block. Voltages and currents are transformed to space phasors and—using the reference angle—rotated to the voltage oriented reference frame. All quantities are fed to the bus (subsection 2.5).

2.2 Power Conversion Models

2.2.1 Power Balance Model

The `PowerBalance` model (see Fig. 3) consists of a three-phase voltage source, prescribing the AC reference voltages of the current controller. The DC current is determined by an integral power controller, comparing AC power and DC power.

The three-phase reference voltages are pre-processed by a block, limiting the amplitude according to the current DC voltage. Additionally, the reference voltages are delayed by half of a switching period $0.5/f_{PWM}$ to consider the dead time delay of a real PWM. The delay is modeled by means of a PT1 for simplicity reasons.

2.2.2 Ideal Switching Model

The `IdealSwitching` model is built from two three-phase IGBT models (`IdealGTOThyristor`) and two anti-parallel three-phase diodes, taken from the Modelica Standard Library.

The switching states of the IGBTs are determined by a space vector PWM block, as described e.g. in [8] and

phi	reference angle of mains voltage
omega	angular velocity of mains voltage
iRef[2]	space phasor of reference current
iAC[2]	space phasor of actual mains current
vAC[2]	space phasor of actual mains voltage
vDC	actual DC voltage
iDC	actual DC current

Table 2: Bus signals

[9]. Additionally the duty cycles are limited according to the current DC voltage at the beginning of each switching period $1/f_{PWM}$.

2.3 MBoxDC

The `MBoxDC` measures DC voltage and current. The current can either be measured between DC capacitor and load terminals, or between converter DC output and DC capacitor terminals. For the latter case, the current drawn by the DC capacitor is added to the measured converter output current to get the desired load current. All measured quantities are fed to the bus (subsection 2.5).

2.4 Current Controller Block

According to subsection 3.2, all components of the current controller have been summarized in this block. The relevant input signals are taken from the control bus (subsection 2.5). The controller output—the three-phase reference voltage—is propagated via a signal output connector.

2.5 Bus

The expandable connector `busMC` contains all signals to be exchanged between measurement models and the different control blocks (Tab. 2). Bus adapters are implemented to connect signal sources for reference values to the bus.

2.6 Parameter Record

This record summarizes all parameters of the ready-to-use models in a convenient way. Additionally, the controller settings (see subsection 3.2 and 3.3) are calculated but can be overwritten by the user, if desired.

2.7 DC Voltage Controller Block

According to subsection 3.3, all components of the DC voltage controller have been summarized in this block.

The desired set point for the DC voltage is defined either as a parameter, or as a signal input. Additionally, the desired reactive reference current can be defined either as a parameter or as a signal input. The controller output—space phasor components of the reference current—are fed to the control bus (subsection 2.5).

3 Controller Design

3.1 Synchronization

Providing a proper synchronization to the base harmonic of the mains voltage even under unbalanced and / or distorted voltage conditions is a crucial task ([10, 11, 12]). Standard solutions use a filter, transform the mains voltages to a space phasor with respect to a static reference frame and rotate this space phasor to the voltage oriented reference frame, using the reference angle that is the output of such a synchronization block. If this rotation is done correctly, the imaginary part of the rotated space phasor should be zero. Thus it could be used as an indicator for a phase locked loop (PLL), both determining the angular velocity and the phase angle of the mains voltage space phasor.

Instead of a PLL, the function `Modelica.Math.atan2` is applied to the voltage space phasor with respect to the static reference frame, thus generating the desired phase angle for synchronization. Detecting the mains frequency or rather angular velocity (i.e. the timely derivative of the phase angle) of the space phasor—especially under unbalanced / and or distorted voltage conditions, even with a filter—is much more complicated. In order to keep the models simple, the constant mains frequency is just fed to the bus (subsection 2.5). Since even the ideal switching power conversion model (subsection 2.2.2) causes voltage harmonics when the grid's stiffness is not infinite, a multi variable filter design ([13, 14]) working on the voltage space phasor with respect to the static reference frame is used (Fig. 4).

Analyzing the filter circuit, we can write

$$\frac{dy}{dt} = k(\underline{u} - \underline{y}) + j\omega_N \underline{y}. \quad (9)$$

For a single harmonic with angular frequency ω the frequency response is

$$\frac{\underline{y}}{\underline{u}} = \frac{k}{k + j(\omega - \omega_N)}. \quad (10)$$

The parameter $k = 2\pi f_B$ defines the bandwidth of the filter. It is obvious that the filter's center frequency

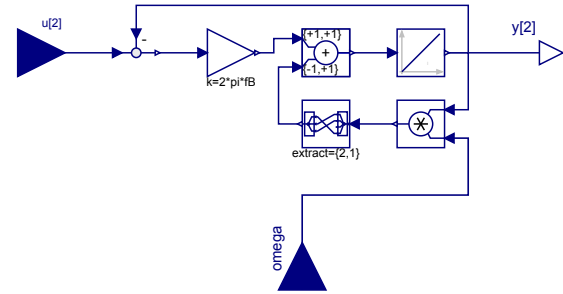


Figure 4: Multi variable filter

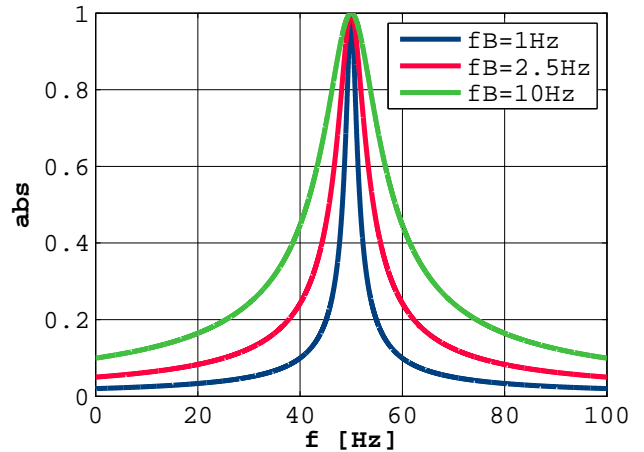


Figure 5: Multi variable filter: gain versus frequency for different bandwidths

ω_N can be adopted simultaneously, e.g. by feeding the current frequency of the PLL. The filter characteristics for various bandwidths are shown in Fig. 5 and 6; at the center frequency gain is 1 and phase shift is 0.

3.2 Current Control Loop

Using the voltage-oriented reference frame, the system can be described using space phasors:

$$\underline{v}_C = \underline{v}_M - R\underline{i} - j\omega L\underline{i} - L\frac{d\underline{i}}{dt} \quad (11)$$

In this equation the index M and C designate the mains and converter side, respectively, and the parameters R and L are the mains resistance and inductance, respectively. Separating the real and imaginary part of this equation and using Laplace transform leads to

$$v_{CRe} = v_{MRe} + \omega L i_{Im} - R(1 + s\tau) i_{Re}, \quad (12)$$

$$v_{CIIm} = v_{MIIm} - \omega L i_{Re} - R(1 + s\tau) i_{Im}, \quad (13)$$

where $\tau = \frac{L}{R}$ is the characteristic time constant.

The complex equation formed by (12) and (13) can be decoupled by means of $\omega L(i_{Im} - j i_{Re})$ such

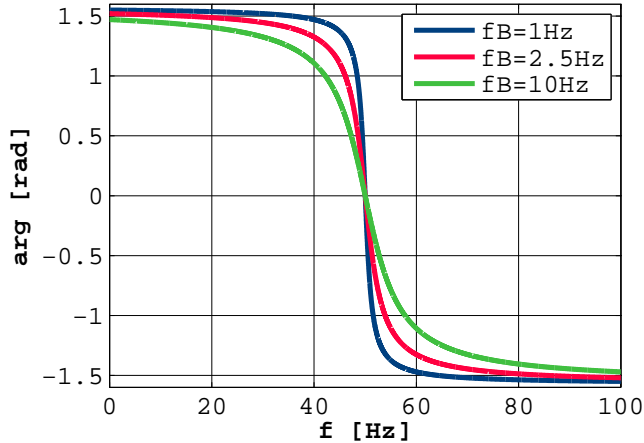


Figure 6: Multi variable filter: phase versus frequency for different bandwidths

that simple PI controllers can be used for controlling the current. Either measured mains voltage ($v_{MRe} + jv_{MIIm}$) can be feed-forwarded, or the integrator states of the PI controllers will meet those values. Compensation of the current's PT1 behavior leads to the controller parameters

$$k_{Current} = k_{DynCurrent} \frac{1}{R}, \quad (14)$$

$$T_{iCurrent} = \tau, \quad (15)$$

where the factor $k_{DynCurrent}$ is used to tune the current controller performance. Neglecting the small dead time delay due to the PWM, the closed current control loop behaves like a PT1 according to

$$i = i_{Ref} \frac{1}{1 + s\tau/k_{DynCurrent}}. \quad (16)$$

The current controller has been investigated with the test example presented in Fig. 7, using the parameters shown in Tab. 1 and a choice of $k_{DynCurrent} = 8$. The gray box emphasizes nearly all components that are included in a ready-to-use model. The DC side of the converter is connected to a constant DC voltage. At time $t = 0.10$ s, a step of the active reference current of 100 Arms (i.e. a real part of the reference space phasor of $\sqrt{2} \cdot 100$ A), and at time $t = 0.15$ s, a step of the reactive reference current of -100 Arms (i.e. imaginary part of the reference space phasor of $\sqrt{2} \cdot 100$ A) is applied.

Fig. 8 and 9 show the real and the imaginary part of the reference and AC current space phasor. Replacing the power balance model `b6` by an ideal switching model, as well as the delay component `pwm` by a space

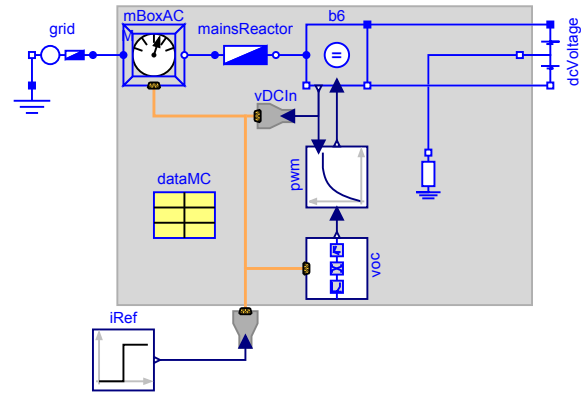


Figure 7: Test example: current control

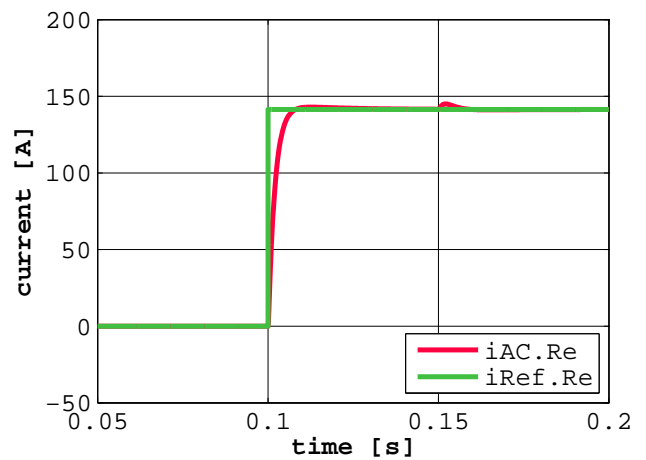


Figure 8: Test example - current control - power balance: real parts of reference current and AC current space phasor

vector PWM block, the results in Fig. 10 and 11 are obtained. The comparison of the power balance and the switching models, however, shows the same qualitative behavior.

3.3 DC Voltage Controller

To describe the relationship between DC voltage, load current and mains current, power balance

$$p_{AC} = \frac{3}{2} v_{MRe} i_{Re} = p_{DC} = v_{DC} i_{DC} \quad (17)$$

can be used, neglecting the losses of the mains inductor (17). Compared with (4), we see that due to the usage of the voltage oriented reference frame, $v_{MIIm} = 0$. Thus we can derive the factor between the real part of the AC current space phasor and the DC

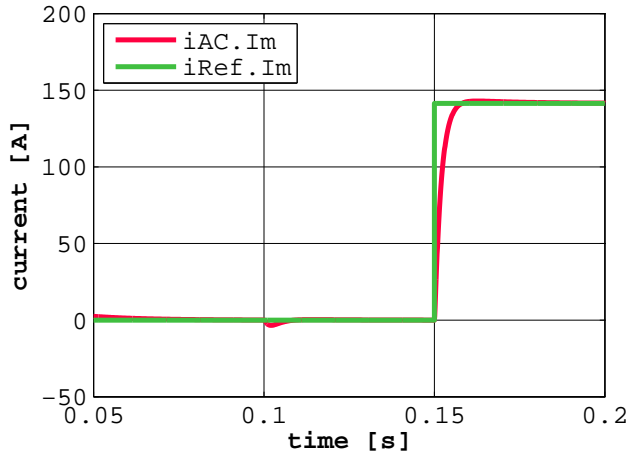


Figure 9: Test example - current control - power balance: imaginary parts of reference current and AC current space phasor

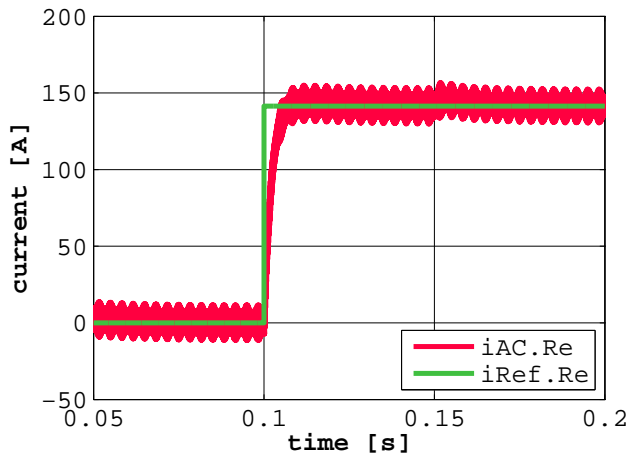


Figure 10: Test example - current control - ideal switching: real parts of reference current and AC current space phasor

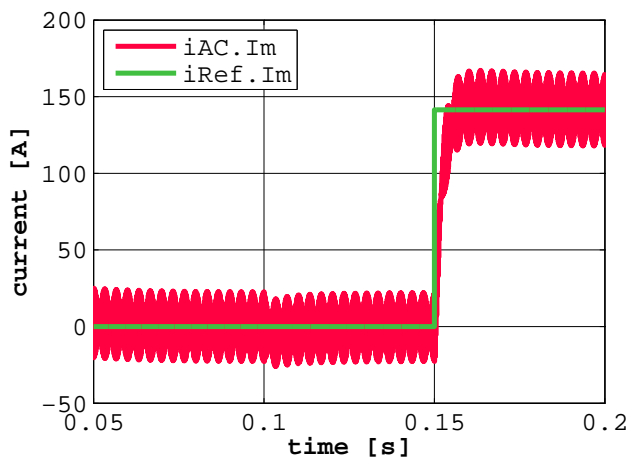


Figure 11: Test example - current control - ideal switching: imaginary parts of reference current and AC current space phasor

load current for stationary operation:

$$k_{ACDC} = \frac{i_{DC}}{i_{AC,Re}} = \sqrt{\frac{3}{2}} \frac{V_{AC,rms,line-to-line,Nominal}}{V_{DC,Nominal}} \quad (18)$$

Using Laplace transform and taking the reference tracking performance of the current control loop (16) into account, we obtain:

$$k_{ACDC} \frac{i_{Ref,Re}}{1 + s\tau/k_{DynCurrent}} = i_{DC,load} + sC_{DC}v_{DC} \quad (19)$$

Using factor $k_{DynVoltage}$ to tune the performance of the DC voltage controller, we choose a PI controller, parametrized according to the symmetrical optimum (standard choice $a = 2$). The gain and time constant of this controller are:

$$k_{Voltage} = k_{DynVoltage} \frac{C_{DC}}{k_{ACDC}} \frac{a}{T_{iVoltage}} \quad (20)$$

$$T_{iVoltage} = a^2 \tau / k_{DynCurrent} \quad (21)$$

The measured load current can be feed-forwarded to enhance the performance of the DC voltage controller. The DC voltage controller has been investigated with the test example presented in Fig. 12, using the parameters shown in Tab. 1, choices of $k_{DynCurrent} = 8$ and $k_{DynDCVoltage} = 2$, and with feed-forward of the measured DC load current. The gray box emphasizes all components that are included in a ready-to-use model. The reference of the DC voltage has been set to nominal DC voltage, the reference of the reactive AC current has been set to zero. At time $t = 0.10$ s, the DC side is loaded with a constant power of $v_{DC,Nominal} \cdot 100$ A.

Fig. 13 shows the DC voltage and Fig. 14 show the real part of the reference and the AC current space phasor. Replacing the power balance model `b6` by an ideal switching model, as well as the delay component `pwm` by a space vector PWM block, the results Fig. 15 and 16 are obtained. The comparison of the power balance and the switching models, however, shows the same qualitative behavior.

4 Complete Example

Finally, the ready-to-use models have been investigated with the test example depicted in Fig. 17, using the parameters shown in Tab. 1, $k_{DynCurrent} = 8$, $k_{DynDCVoltage} = 2$, and without feed-forward of the measured DC load current. A DC voltage controller

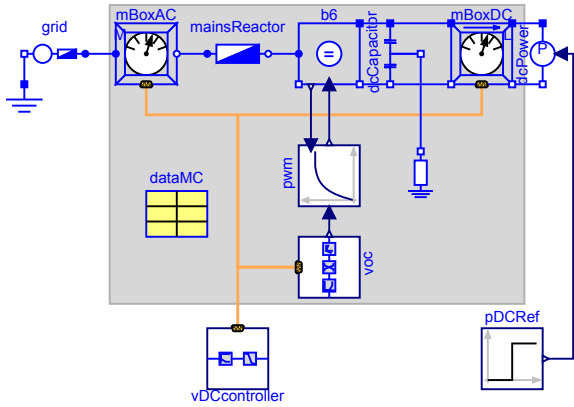


Figure 12: Test example: DC voltage control

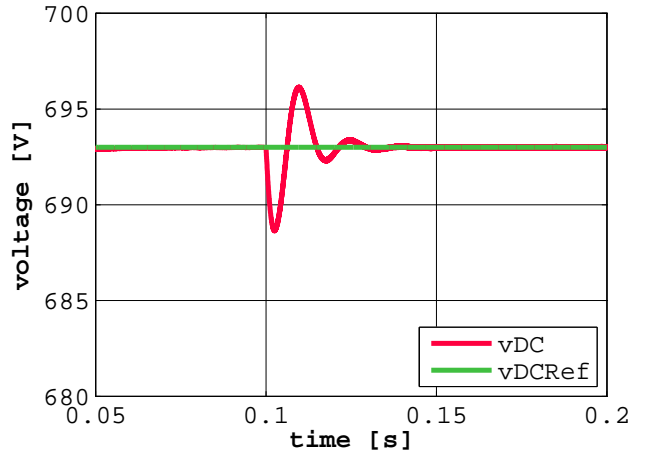


Figure 15: Test example - DC voltage control - ideal switching: reference and actual DC voltage

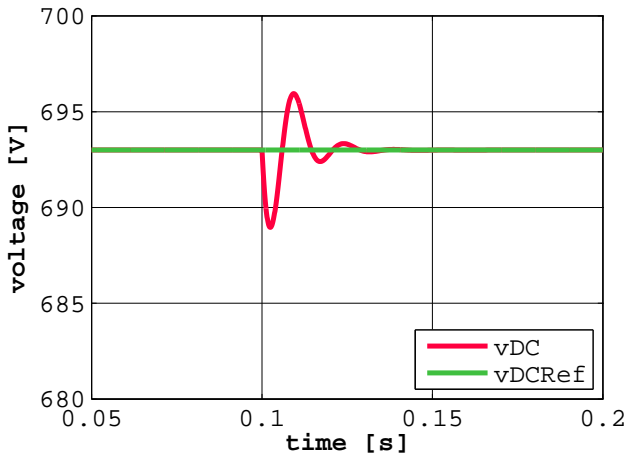


Figure 13: Test example - DC voltage control - power balance: reference and actual DC voltage

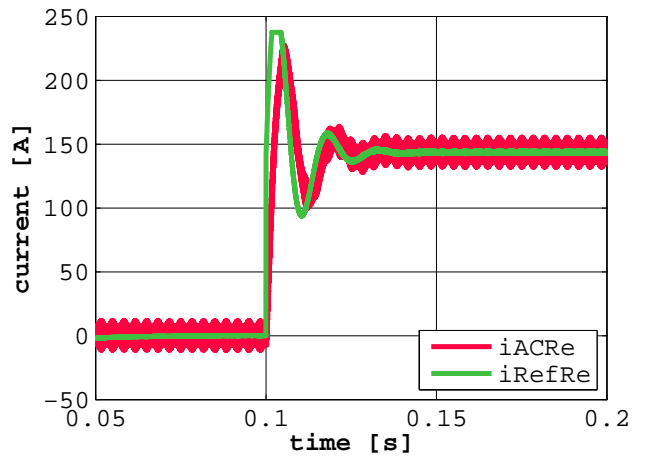


Figure 16: Test example - DC voltage control - ideal switching: real parts of reference current and AC current space phasor

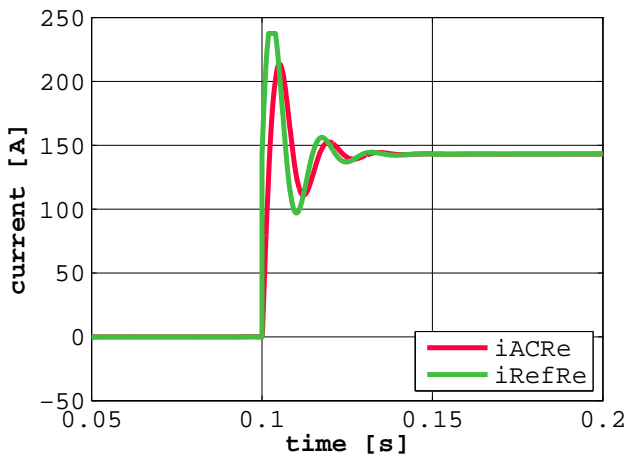


Figure 14: Test example - DC voltage control - power balance: real parts of reference current and AC current space phasor

is connected to mains converter $mc1$; mains converter $mc2$ controls active and reactive current, thus creating a DC load for mains converter $mc1$. At time $t = 0.10 s$, a step of the active reference current of $100 A$ (i.e. a real part of the reference space phasor of $\sqrt{2} \cdot 100 A$) is applied to the mains converter $mc2$; the set point of reactive reference current remains 0. The AC terminals of both mains converters are connected to the same mains, thus the grid delivers only active current to cover losses (of the mains reactors). Since both converter models contain a DC capacitor (which is initialized at nominal DC voltage), the DC terminals have to be connected via small bus bar resistances.

Fig.18 shows real parts of reference and AC current space phasor of mains converter $mc2$. The resulting real parts of reference and AC current space phasor of

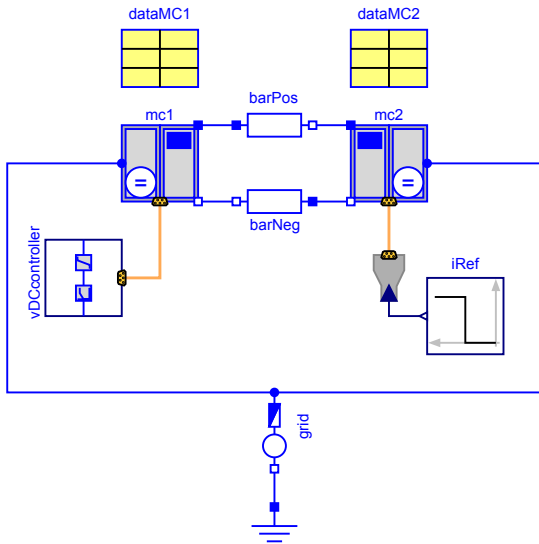


Figure 17: Complete example

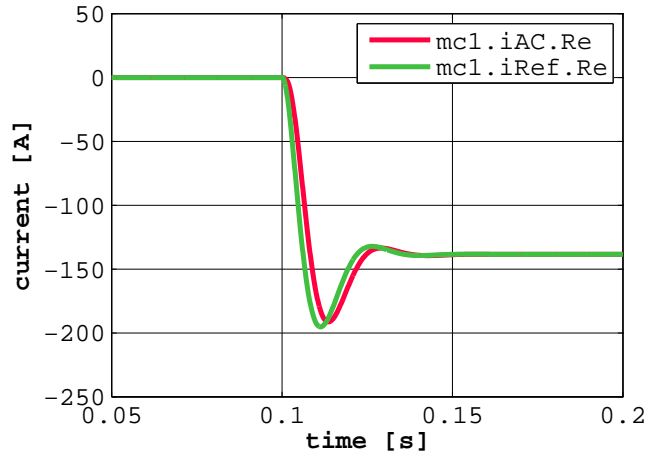


Figure 19: Complete example - power balance, Converter1: real parts of reference current and AC current space phasor

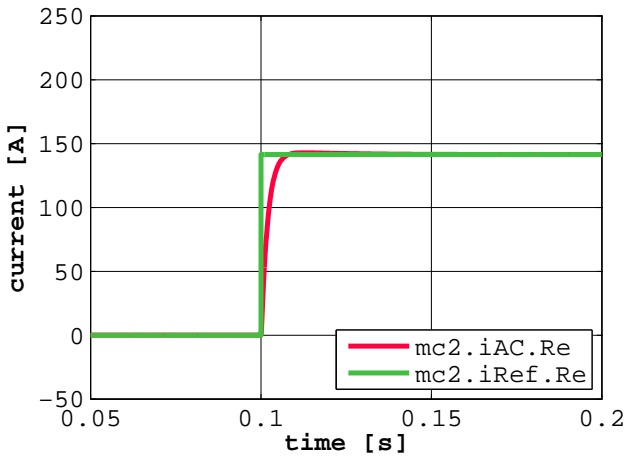


Figure 18: Complete example - power balance, Converter2: real parts of reference current and AC current space phasor

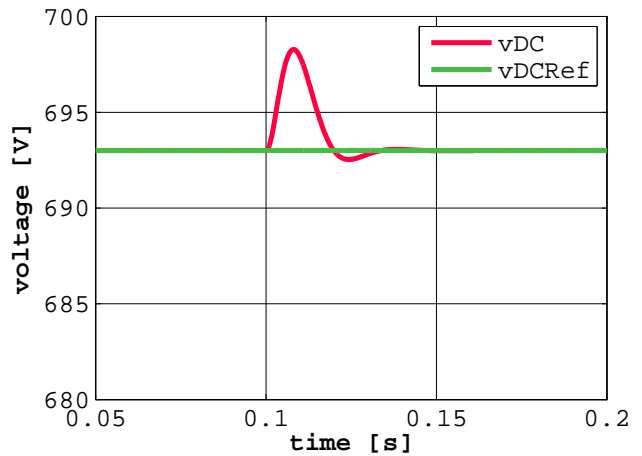


Figure 20: Complete example - power balance: reference and actual DC voltage

mains converter $mc1$ are depicted in Fig. 19. DC voltage is controlled very well, as demonstrated in Fig. 20. Replacing the power balance models $mc1$ and $mc2$ by ideal switching models, the results Fig. 21–23 have been obtained. The comparison of the power balance and the switching models, however, shows the same qualitative behavior.

5 Conclusions

All components needed to implement ready-to-use models of an AFE, i.e. a mains connected PWM converter with voltage oriented control, have been presented and tested by means of simulations. A complete example demonstrates the usage of the ready-to-use

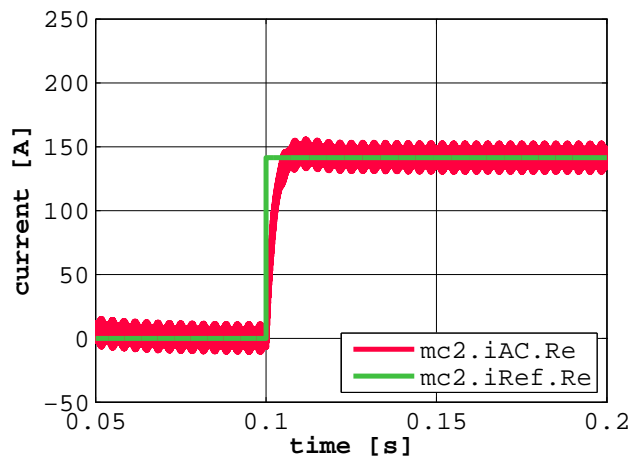


Figure 21: Complete example - ideal switching, Converter2: real parts of reference current and AC current space phasor

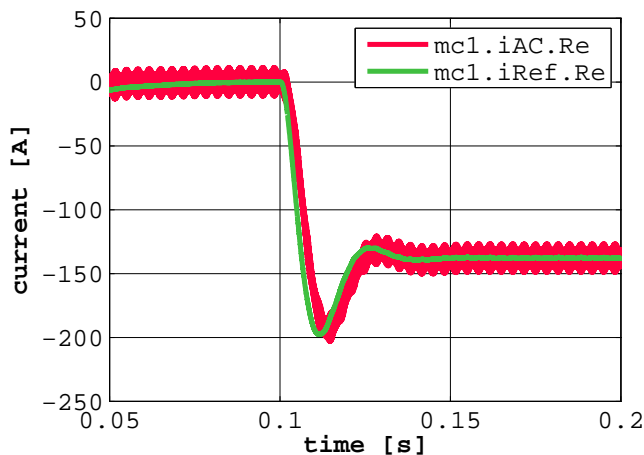


Figure 22: Complete example - ideal switching, Converter1: real parts of reference current and AC current space phasor

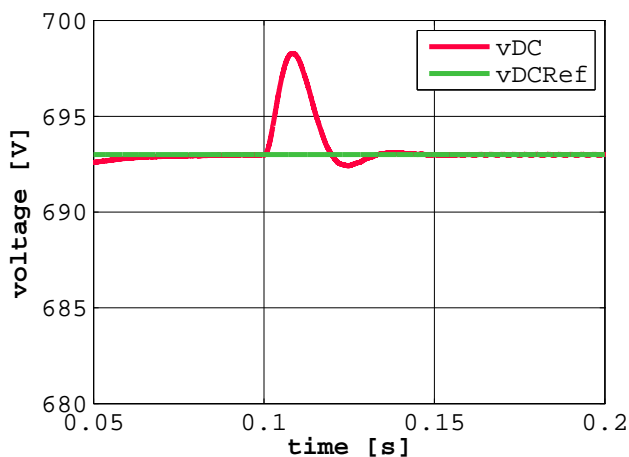


Figure 23: Complete example - ideal switching: reference and actual DC voltage

models. The user can choose between a power balance model and an ideal switching model. As expected, the power balance model—neglecting switching effects—speeds up simulation by a factor of more than 100, of course dependent on the PWM switching frequency. Using these AFE models, a lot of investigations can be carried out, like:

- Behavior of the AFE under unbalanced or distorted mains voltage
- Parallel operation of AFEs at the same DC bus
- Stability of an AFE under weak grid conditions or connected to a synchronous generator
- Implementation of power controllers for wind turbines or solar converters

- Usage of an AFE as reactive power compensation

Further development is planned for:

- Coupling the AFE models with drive models from the `SmartElectricDrives` library
- Enhancing the synchronization
- Adapting current control to mains-side LCL filters

References

- [1] J. V. Gragger, H. Giuliani, C. Kral, T. Bäuml, H. Kapeller, and F. Pirker, “The SmartElectricDrives Library – powerful models for fast simulations of electric drives,” *The 5th International Modelica Conference*, pp. 571–577, 2006.
- [2] M. Malinowski, M. Kazmierkowski, and A. Trzynadlowski, “Review and comparative study of control techniques for three-phase pwm rectifiers,” *Mathematics and Computers in Simulation*, vol. 63, p. 349–361, 2003.
- [3] M. Malinowski, M. P. Kazmierkowski, and A. M. Trzynadlowski, “A comparative study of control techniques for pwm rectifiers in ac adjustable speed drives,” *IEEE Transactions On Power Electronics*, vol. 18, no. 5, pp. 1390–1396, May 2003.
- [4] S. L. Sanjuan, “Voltage oriented control of three-phase boost pwm converters,” Master’s thesis, Chalmers University of Technology, Gothenburg, 2010.
- [5] F. Blaabjerg, R. Teodorescu, Z. Chen, and M. Liserre, “Power converters and control of renewable energy systems,” *International Conference on Power Electronics, ICPE 2004*, pp. 2–20, 2004.
- [6] T. Østrem, “Reliable electric power conversion for connecting renewables to the distribution network,” Ph.D. dissertation, Norwegian University of Science and Technology, 2008.
- [7] S. E. Evju, “Fundamentals of grid connected photovoltaic power electronic converter design,” Master’s thesis, Norwegian University of Science and Technology, 2007.
- [8] J. V. Gragger, A. Haumer, C. Kral, and F. Pirker, “Efficient analysis of harmonic losses in PWM voltage source induction machine drives with Modelica,” *International Modelica Conference, 6th, Bielefeld, Germany*, pp. 593–600, 2008.
- [9] A. Boglietti, G. Griva, M. Pastorelli, F. Profumo, and T. Adam, “Different PWM modulation techniques indexes performance evaluation,” *IEEE International Symposium on Industrial Electronics, ISIE’93 - Budapest.*, pp. 193–199, 1993.

- [10] F. D. Freijedo, J. Doval-Gandoy, O. Lopez, C. Martinez-Penalver, A. G. Yepes, P. Fernandez-Comesana, J. Malvar, A. Nogueiras, J. Marcos, and A. Lago, "Grid-synchronization methods for power converters," *35th Annual Conference of IEEE Industrial Electronics, IECON 2009*, pp. 522–529, 2009.
- [11] A. Nagliero, R. A. Mastromauro, M. Liserre, and A. Dell'Aquila, "Synchronization techniques for grid connected wind turbines," *35th Annual Conference of IEEE Industrial Electronics, IECON 2009.*, pp. 4606–4613, 2009.
- [12] A. Timbus, R. Teodorescu, F. Blaabjerg, and M. Liserre, "Synchronization methods for three phase distributed power generation systems. an overview and evaluation," *IEEE 36th Power Electronics Specialists Conference, PESC 2005.*, pp. 2474 – 2481, 2005.
- [13] T. Østrem, W. Sulkowski, L. E. Norum, and C. Wang, "Grid connected photovoltaic (pv) inverter with robust phase-locked loop (pll)," *IEEE PES Transmission and Distribution Conference and Exposition Latin America*, 2006.
- [14] M. C. Benhabib, F. Wang, and J. L. Duarte, "Improved robust phase locked loop for utility grid applications," *13th European Conference on Power Electronics and Applications, EPE 2009*, pp. 1–8, 2009.

A real time capable battery model for electric mobility applications using optimal estimation methods

Jonathan Brembeck

Sebastian Wielgos

German Aerospace Center (DLR) Oberpfaffenhofen, Institute of Robotics and Mechatronics

Muenchner Strasse 20, D-82234 Wessling, Germany

jonathan.brembeck@dlr.de Sebastian.Wielgos@schaeffler.com

Abstract

In this paper a modeling approach for a Lithium-Ion cell online monitoring and offline benchmarking is proposed. It combines physical modeling in equivalent electric circuit representation with grid tables of cell type characteristic information from laboratory tests. The model is fully parameterized and validated with cells used in the *HighVoltage* battery pack of DLR research robotic electric vehicle ROboMObil.

Keywords: BMS (BatterieManagementSystem); SOC (StateOfCharge); EKF (ExtendedKalmanFilter); BEV (BatteryElectricVehicle);

1 Introduction

Nowadays electric mobility gets more and more important. New manufacturing methods give the hope that in the near future vehicles for individual transport can be fully electrified. The use of secondary chemical energy storages is one of the main development tasks in the automotive industry. Besides of the development of new chemical mixtures for higher power density and durability, it is also necessary to develop new embedded systems and advanced algorithms for battery management systems and global energy distribution strategies. The aim of these systems is to give a good estimation for actual and future power availability and health monitoring. This requirement is very complex due to the nonlinear behavior, especially with high performance Lithium-Ion cells. Currently no direct measurement method without destruction of the cell is available to determine the major characteristic parameters and states. In this paper a model for use with recursive online estimation is suggested which gives a good trade-off between

modeling accuracy and real-time requirements like low order system or a minimal number of (non)linear subsystems. The proposed algorithm is successfully implemented, parameterized and tested within the ROboMObil project (Figure 1, [Bre11]), a research platform for future electric mobility developed at the DLR Institute for Robotics and Mechatronics.



Figure 1: ROboMObil test drive

The suggested model is also integrated in the *BatteryElectricVehicle* model for drive cycle simulations of the overall system in [Eng10]. It is there applied for offline analysis and parameter variation.

2 Different approaches for cell modeling

2.1 Models for offline purposes

To model the electric behavior of a cell, the obvious thing to do is to use an equivalent circuit. As an example for this type of modeling, the approach from [Böh08] is presented. The cell behavior is separated into three time domains. The impedance

determines the short-time range while the long-term behavior is incorporated by a voltage source. Finally, the transitional behavior is captured by a number of exponential functions which are overlaid.

Another possibility for cell modeling is based upon impedance spectroscopy measurements. An example for this class of models can be found in [Sti08] with its enhanced equivalent circuit. The level of detail is significantly higher compared to [Böh08] due to the continuous formulation from low to high frequency effects. The following cell characteristics are considered: ohmic resistances, parasitic inductances, charge transfer and double layer capacity, diffusion processes and formation of solid electrical interface.

For purposes of cell design, the Comsol *Batteries and Fuel Cells Module* ([Com10]) is an advanced modeling method. The model is so accurate that it is possible to simulate the concentration of the electrolyte and therefore enables battery engineers to test different combinations of materials and dimensions to optimize the cell behavior.

2.2 Models for online purposes

The presented offline models are not applicable to embedded control systems due to their modeling approaches and the relatively complicated representations. Available *BatterieManagement Systems*, e. g. the widely used system from *I+ME Actia*, use a predetermined cell characteristic table and a current counting method without considering the transient behavior of the cell.

Another and more advanced approach is the *EnhancedSelfCorrecting* model by Plett proposed in [Ple04], [Ple04b], [Ple04c] and [Ple04d]. The cell is considered as a system with input "cell current" and output "terminal voltage". The *StateOfCharge* l is included in the state vector and therefore can be estimated by means of an *ExtendedKalmanFilter*. The basis of the model is the *OpenCircuitVoltage* U_{OCV} and the ohmic loss. This is represented in the equivalent circuit in Figure 2.

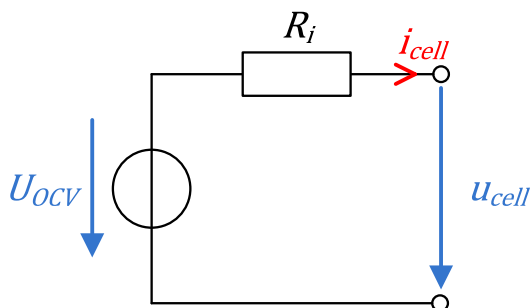


Figure 2: Equivalent circuit representation

Beyond that, this approach takes into account hysteresis effects. The remaining cell dynamics is described by means of a current filter. Therefore, this results in the following output equation:

$$u_{cell} = U_{OCV}(l) + h - R_i \cdot i_{cell} + u_f \quad (1)$$

where h is the hysteresis voltage and u_f represents the influence of the current filter.

The implementation of the ESC model shows significant optimization potential. There are several better discretization methods than the proposed explicit *Euler 1 method*. This helps in simulation stability and accuracy for online and offline purpose (see section 3.3 for details). The formulation of the current filter also seems unnecessary complex, It is formulated as an IIR Low Pass Filter as follows: $LowPass = 1 - HighPass$.

Furthermore the computational costly online parameterization of the filter by the use of a dual estimation approach (see [Ple04d]) can be done offline. In this way the system order can be reduced and therefore the online performance increases. Moreover, the first implementations of the ESC model at DLR have shown problems with the determination of *SOC*. In addition, the effects of current and temperature on the actual cell capacity are not considered in the ESC model. These can have an enormous influence on the calculation of the *SOC*, cf. Figure 3.

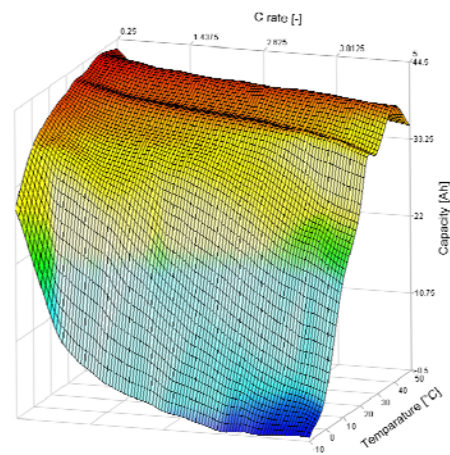


Figure 3: Capacity variation due to temperature and current ($C_{Rate} = C_N/I$)

2.3 Proposal of the modifiedESC model

Since we are convinced of the basic approach with the *ESC* model, we will introduce a modified version that is different in several aspects. In the *mESC* approach, the battery cell is represented by a continuous state space model. Through this scheme, the discretization algorithm can be changed and methods with higher accuracy, such as Runge-Kutta 4, can be used. Due to the complex structure and time consuming parameterization of the current filter, it is replaced by a critical damping FIR filter that only contains one parameter. Additionally, the calculation of the *SOC* is improved by an empirically determined correction factor that accounts for the dependency of temperature and current on the actual cell capacity. Lastly, the model is parameterized using an offline optimization with real training data and then verified by validation data.

3 mESC model details

3.1 Derivation and model structure

The mathematical description of the *mESC* model is given by the following:

$$\begin{bmatrix} \dot{l} \\ h \\ f_f \end{bmatrix} = \begin{bmatrix} -\frac{\eta_{Ah} \cdot k_i}{C_N} \cdot i_{cell} \\ \left| \frac{\gamma \cdot \eta_{Ah} \cdot k_i}{C_N} \cdot i_{cell} \right| \cdot (M - h) \\ -\omega \cdot f_{f1} + \omega \cdot i_{cell} \\ \omega \cdot f_{f1} - \omega \cdot f_{f2} \\ \omega \cdot f_{f2} - \omega \cdot f_{f3} \\ \omega \cdot f_{f3} - \omega \cdot f_{f4} \end{bmatrix} \quad (2)$$

$$u_{cell} = U_{OCV}(l) + h - R_i \cdot f_{f4} \quad (3)$$

The differential equation for the *SOC* l depends on the cell current i_{cell} , the nominal cell capacity C_N , the Coulombic efficiency η_{Ah} and the correction factor k_i . This factor takes into consideration the variation of the cell capacity as shown in Figure 3. Following [Gra02] the correction factor is determined by:

$$k_i = \begin{cases} c_i \cdot i_{cell} + k_0 & \text{for } i_{cell} > 0 \text{ (chg.)} \\ e^{c_i \cdot i_{cell}} + (k_0 - 1) & \text{for } i_{cell} < 0 \text{ (dchg.)} \end{cases} \quad (4)$$

c_i is a positive constant leading into a straight line for positive cell current, which intersects the ordinate at k_0 . For negative cell current the

correction factor is described by an exponential function. The parameters c_i and k_0 are gathered from capacity tests replacing the simple straight line with more accurate look-up tables.

The hysteresis voltage h is described by a more complex equation taking into account the additional factors M (polarization voltage) and γ (time constant). The remaining four differential equations describe the optimized fourth order critical damping current filter with the only remaining parameter ω and its four states f_{fx} .

The output equation (3) is similar to the *ESC* model's output equation but aggregates the influence of the cell current (ohmic loss and current filter) into one summand.

3.2 Important variables and parameters

The internal resistance is one of the cell's descriptive variables, which depends on *SOC*, cell current and temperature resulting in a three dimensional look-up table. This relationship is visualized for room temperature in Figure 4. Considering equation (3) one can imagine easily that the resulting cell voltage varies highly in case of low *SOC* and high currents due to increase of the internal resistance of the cell. In such cases the cell is in a critical situation and can be damaged irreversibly.

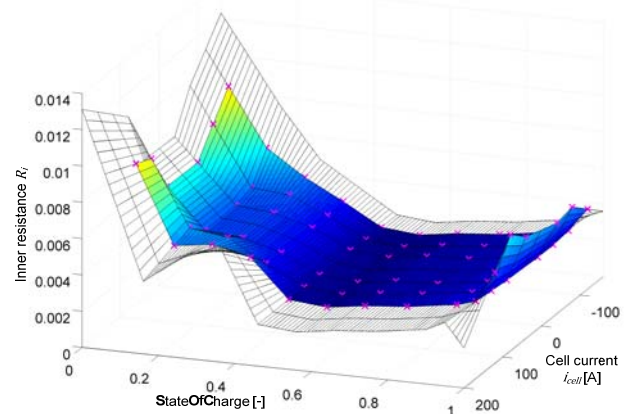


Figure 4: Internal cell resistance for $T = 25^\circ\text{C}$

Another important cell variable is the *OpenCircuitVoltage*, whose characteristic curve incorporates the relationship between *SOC* and *OCV*, as shown in Figure 5. This depiction also shows the hysteresis effects very well. The blue and red curves are measured during charging and discharging of the cell with very low currents respectively. This minimizes excitation of the cell dynamics so that the cell terminal voltage can be considered unloaded. In addition, the influence of the internal resistance is eliminated during the data

analysis. The polarization voltage is defined as half of the difference between the two curves and therefore also depends on the OCV.

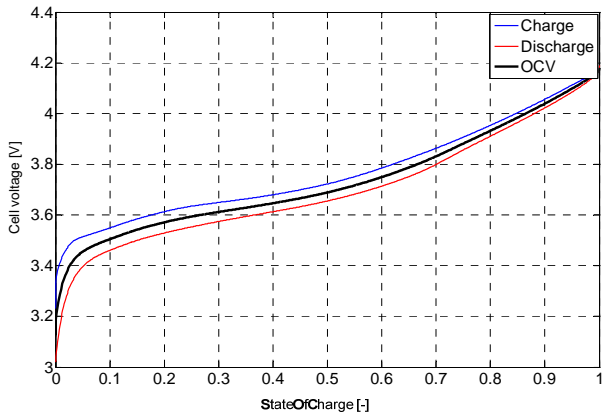


Figure 5: OpenCircuitVoltage

In Figure 6 the polarization voltage is plotted depending on the temperature. It is apparent that temperatures below - have the highest influence.

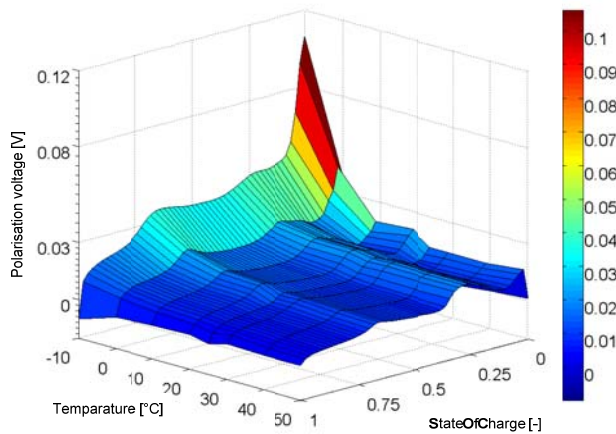


Figure 6: Polarization voltage

Finally, Figure 7 shows the dependency of an exemplary correction factor on i_{cell} at room temperature. Measurement data is collected only for discharge and stored in a look-up table. For charging the data is calculated using equation (4).

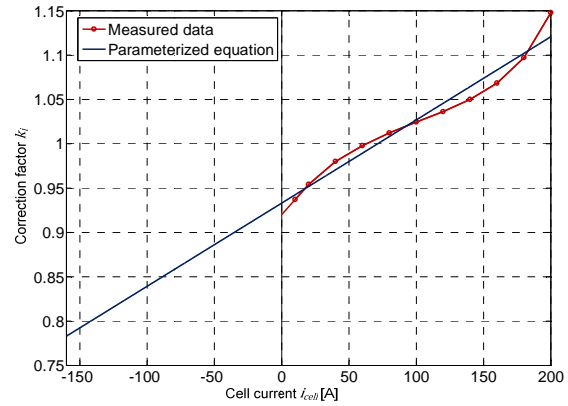


Figure 7: Exemplary identification of k_i for $T=25^\circ\text{C}$

The discussed model is implemented in Modelica as part of DLRs ROMO Energetic library. It is used for offline simulations to optimize the energy management strategies of the vehicle controllers and implemented in the central control unit of the ROboMObil using a rapid prototyping environment. The resulting implementation in DYMOLA is depicted as follows.

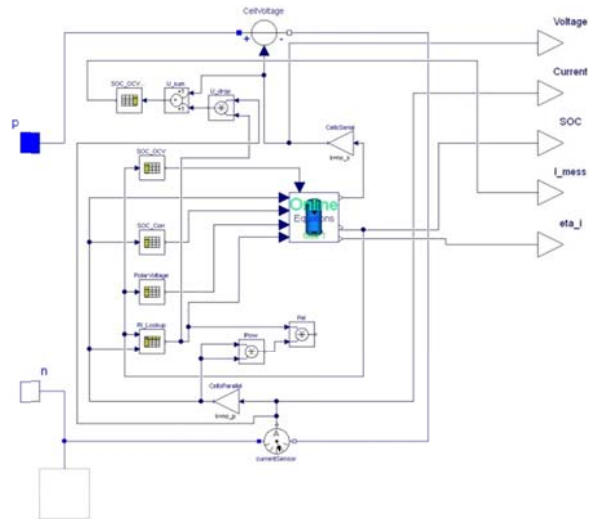


Figure 8: mESC model implementation in DYMOLA

3.3 mESC model and Kalman filtering

The proposed mESC model has one input, cell current, and one output, cell voltage. These two quantities can be directly measured with high accuracy even in embedded systems. The third quantity is the cell temperature. It is determined by the use of a thermocouple sensor on the cell surface. This is the same method as applied in the cell test bench (see section 4 for details). To achieve a better estimation performance, a second measurement equation is implemented. The idea in principal is to take constraints into account with a recursive Kalman Filter. For this purpose an additional

fictitious measurement is introduced. It can be weighted through the tuning of the output covariance matrix. This method is well known as *perfect measurements* ([Sim06] chp. 7.5.2). In this way the output equation extends to:

$$y = \begin{bmatrix} u_{cell} \\ l \end{bmatrix} \quad (5)$$

The first equation is identical to formula (3) and the second one can be derived as follows:

$$\begin{aligned} u_{cell} &\approx U_{OCV}(l) - R_i \cdot i_{cell} \\ \Rightarrow U_{OCV}(l) &\approx u_{cell} + R_i \cdot i_{cell} \\ \Rightarrow l &\approx l_{meas} = U_{OCV}^{-1}(u_{cell} + R_i \cdot i_{cell}) \end{aligned} \quad (6)$$

The measured SOC is calculated through the inverse OCV/ U_{OCV}^{-1} look-up in combination with a low pass filtering afterwards. This extension allows the *ExtendedKalmanFilter* to adjust SOC directly and therefore to enforce a physically correct estimation. To use the proposed mESC model in an embedded system for state estimation an EKF is implemented. It handles the nonlinearities through linearization in each time step. In Table 1 the algorithm for discrete systems is shown as found in standard literature (e. g. [Sim06]).

Table 1: *Extended Kalman Filter Algorithm*

Initialization $\hat{x}_0 = E(x_0)$ $P_0^+ = E[(x_0 - \hat{x}_0^+)(x_0 - \hat{x}_0^+)^T]$ For $k = 1, 2, \dots$ $\hat{x}_k^- = f_{k-1}(\hat{x}_{k-1}^+, u_{k-1})$ $P_k^- = \Phi_{k-1} P_{k-1}^+ \Phi_{k-1}^T + Q$ where $\Phi_{k-1} = \left. \frac{\partial f_{k-1}}{\partial x} \right _{\hat{x}_{k-1}^+}$ $K_k = P_k^- H_k^T \cdot (H_k P_k^- H_k^T + R)^{-1}$ where $H_k = \left. \frac{\partial h_k}{\partial x} \right _{\hat{x}_k^-}$ $\hat{x}_k^+ = \hat{x}_k^- + K_k \cdot (y_k - h_k(\hat{x}_k^-))$ $P_k^+ = (I - K_k \cdot H_k) \cdot P_k^-$
--

Through the fact that our estimation model (equations (2) and (5)) is in a continuous state space formulation, it is necessary to discretize the model in each time step. This can be done by several methods. The easiest way is by the use of an Euler1 method. Because of the poor stability (all poles have to be placed within the unit circle of the time dependent complex plane) we suggest using a Trapezoid method (eq. (7)). It guarantees that the prediction step (\hat{x}_k^-) is always stable, even for a large sampling time T_s , as long as all poles of the continuous system stay in the left half of the complex plane.

$$\frac{\dot{x}(t_k) + \dot{x}(t_k + T_s)}{2} = \frac{x(t_k + T_s) - x(t_k)}{T_s} \quad (7)$$

After some calculations the prediction step is described as follows,

$$\hat{x}_k^- = \hat{x}_{k-1}^+ + \left(I - \frac{T_s}{2} F_{k-1} \right)^{-1} \cdot T_s f_k \quad (8)$$

where F_{k-1} is the Jacobi matrix of the continuous system with state vector $k-1$. To receive the transition matrix Φ_{k-1} we have to solve the following equation:

$$\Phi_{k-1} = \left(I - \frac{T_s}{2} F_{k-1} \right)^{-1} \cdot \left(I + \frac{T_s}{2} F_{k-1} \right) \quad (9)$$

For reason of numerical stability all matrix inversions are done by solving a linear equation problem of the form $A \cdot x = b$. This can be done with *LU_solve2* from *ModelicaStandard*. In conclusion we have an *Extended Kalman Filter* algorithm which has $O(nx^3)$ -flops more than the original filter, but outperforms it due to the fact that the sampling time can be chosen much larger. This is only limited by Shannon's sample theorem for sampling the measured signals in real-time. Another limit is that the linearization of the EKF causes a mean and covariance propagation which is only valid to the first order. This is due to Taylor series expansion being truncated after the first term. The second issue can be improved by the use of higher order methods like the *Unscented Kalman Filter*, [Mer04].

4 Parameterization and validation

In the ROboMObil project we were able to obtain a high performance cell from *Li-Tec industries*. It has a nominal capacity of 40 Ah and with its security features it is fully capable for series production. All cell measurement for parameterization, testing, and validations were done with a *Vötsch VT4011* environment simulator and a *BaSyTec* battery testing system. The model parameter optimization is accomplished with MOPS (see [Joo08] for more information) on the Linux cluster of DLR RM Institute. The detailed procedure and the necessary test cycles are explained in [Wie10]. The list of tuned parameters is given in Table 2.

Table 2: List of optimized parameters

γ	Time constant for rate of change of the hysteresis voltage
f_g	Cut-off frequency of the fourth order current filter with critical damping
f_l	Cut-off frequency of the first order low pass filter for smoothing the "measured" SOC needed due to the extended output equation
Q, R	Covariance matrices for optimal EKF settings (weighting of prediction by means of the model and correction due to the measured values)

The optimization process can be summarized as shown in Figure 9. As quality criterions cell voltage and SOC are compared to their respective references using the so-called fit value. The fit value weights the reference to the simulated characteristic vector and can be calculated as follows:

$$Fit = \left(1 - \frac{\sqrt{\sum_{i=1}^n (|y_{h,i} - y_i|^2)}}{\sqrt{\sum_{i=1}^n (|y_i - \frac{1}{n} \cdot \sum_{i=1}^n y_i|^2)}} \right) \cdot 100 \quad (10)$$

Since a direct comparison of SOC is not possible, the actually and effectively moved amount of charge at the terminals of the cell is used.

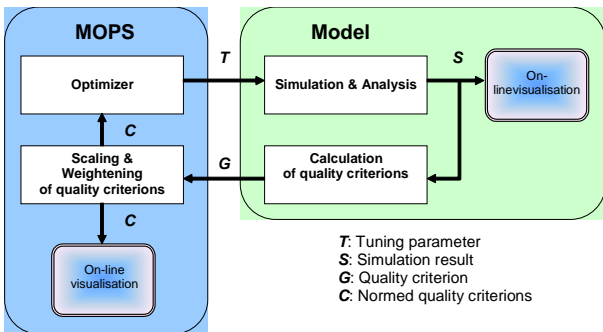


Figure 9: MOPS optimization process

For the validation of the estimated model parameters a testbench experiment by means of a simulated drive cycle is used. The observer scheme is shown in Figure 10. The aim is to produce an accurate estimate of the battery *StateOfCharge*, which is a decisive input for the function of the energy management strategy. In this observer, the input u of the battery model is the measured current, while the model output y_m is the voltage of a single cell.

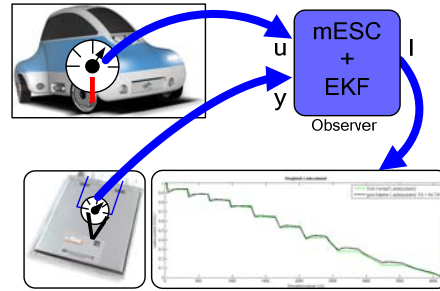


Figure 10: mESC observer structure

The model used in the observer was parameterized by an offline optimization that uses measurements test cycles and the characteristics of a single cell as training data. In order to validate the resulting parameterization, the battery was connected to an electrical power supply/load and tested with a simulated drive cycle. For this simulation, a model of the longitudinal dynamics of the ROboMObil was developed in Modelica which calculates the energy flow of all the power consumers in the electrical system [Eng10].

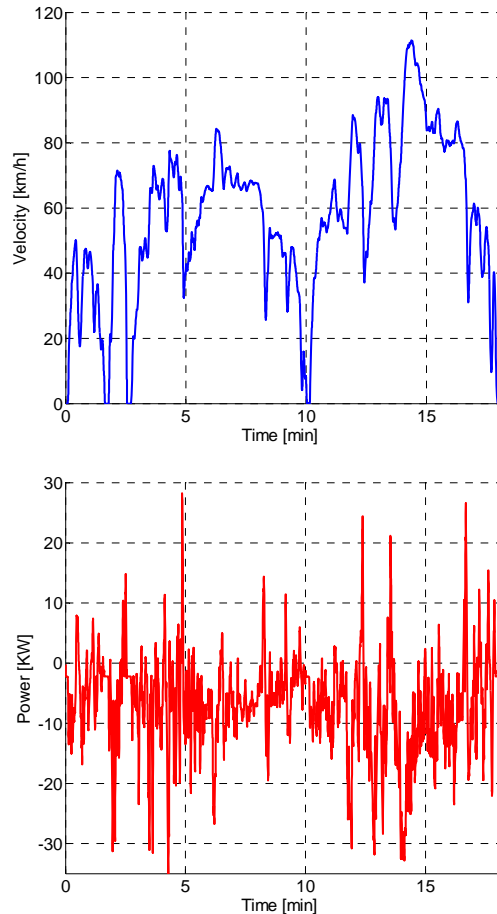


Figure 11: Artemis Road velocity profile and power consumption

A closed-loop controller calculates the actuator demands in order to follow the specified driving cycle. In this way, the required electrical power is calculated. Figure 12 shows a normalized Artemis Road cycle, which is synthesized by stochastic calculations from real recorded driving data [And04]. This reflects real driving behavior significantly better than the purely synthetic ECE15 driving cycle used in the homologation of European vehicles. The calculated electrical power flow is then taken from a Dymola simulation and scaled to one battery cell. This data is used on the HIL test bench to give the Lithium-Ion cell the appropriate load current. The results of this test, including the current, voltage, as well as the cell temperature are used to test the mESC model with EKF in the offline simulation. The test results are shown in Figure 12, where the measured and estimated values are compared to each other. Evidently, the voltage values agree very well. With almost 93% accuracy, the accordance of the effective charge amount delivers a good result and shows that the selected approach with the model-based observer is of practical use.

5 Conclusions and future work

We have introduced a modeling approach for Lithium-Ion cells that shows good performance and was implemented within a practical application (ROboMObil). Future development will extend the model with real-time capable temperature dynamics as suggested in [Che09], [Mi07] and [Mat08]. This should lead to better results in prediction of power availability during critical situations like sub-zero temperatures. Furthermore validation tests on the rapid prototyping embedded systems in the ROboMObil are planned. In the first weeks of

January 2011 we are able to accomplish roller bench experiments with the ROboMObil. We are looking forward to gaining new insights from the recorded measurement data of the energy system. The results will be presented in an upcoming publication.

6 Acknowledgment

We would like to thank Prof. M. Otter for his support regarding implementation. His extensive knowledge of robust and reliable numerical matrix calculus was instrumental in achieving these good results. Furthermore we would like to thank LionSmart GmbH for the open discussions in points of model implementation.

References

- [And04] André, M. (2004). **The ARTEMIS European driving cycles for measuring car pollutant emissions.** *Science of The Total Environment*, 334-335, 73-84.
- [Com10] **Batteries and Fuel Cells Module.** Retrieved 2010, from <http://www.comsol.com/showroom/gallery/686/>
- [Böh08] Böhm, K. A. (2008). **Charakterisierung und Modellierung von elektrischen Energiespeichern für das Kfz.** Aachen: Shaker Verlag.
- [Bre11] Brembeck, J., Ho, L. M., Schaub, A., Satzger, C., & Hirzinger, G. (2011). **ROMO – the robotic electric vehicle** - Submitted for publication. *IAVSD*
- [Eng10] Engst, C., Brembeck, J., Otter, M., & Kennel, R. (2010). **Object-Oriented Modelling and Real-Time Simulation of an Electric Vehicle in Modelica.** Technische Universität München

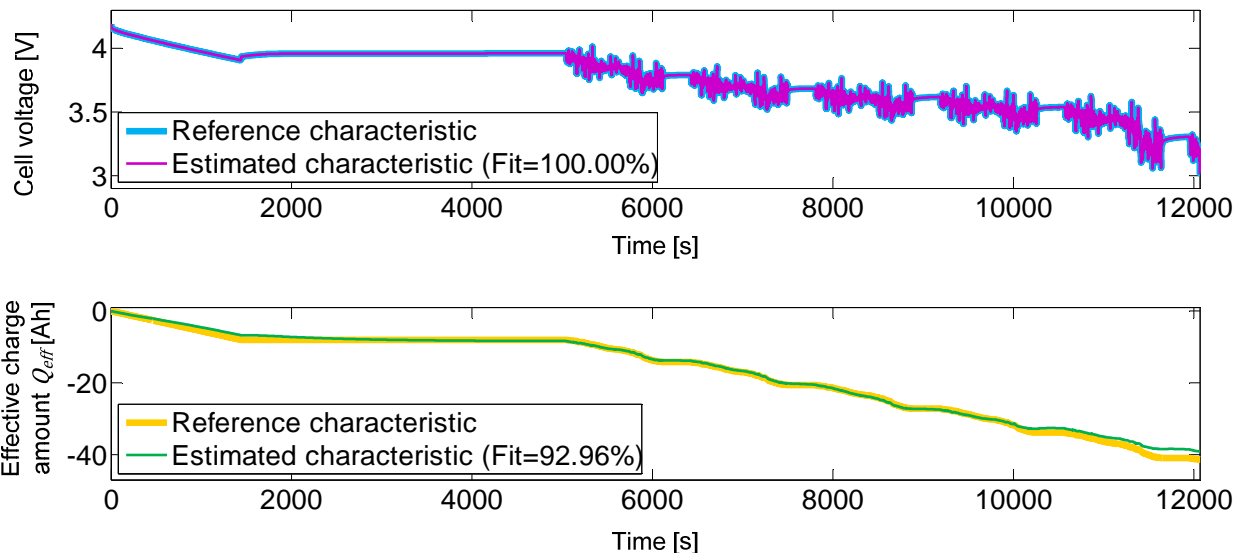


Figure 12: Experiment results from Artemis Road Drive Cycle Test

- [Che09] Cheng, L., Ke, C., Fengchun, S., Peng, T., & Hongwei, Z. (2009). **Research on thermo-physical properties identification and thermal analysis of EV Li-ion battery.**, *Vehicle Power and Propulsion Conference, 2009. VPPC '09. IEEE*, 2009, 1643 -1648
- [Gra02] Graaf, R. (2010). **Simulation hybrider Antriebskonzepte mit Kurzzeitspeicher für Kraftfahrzeuge.** Ika RWTH, Aachen.
- [iME10] i+ME ACTIA. (n.d.). Retrieved 11 2010, from http://www.ime-actia.com/download/IR11652_BMS_GB.pdf
- [Joo08] Joos, H.-D., Bals, J., Looye, G., Schnepfer, K., & Varga, A. (2008). **MOPS: Eine integrierte optimierungsbasierte Entwurfsumgebung für mehrzielige, parametrische Analyse und Synthese.** DGLR Workshop Systemidentifizierung, Parameterschätzung und Optimierung.
- [Lio10] LionSmart. (2010). Retrieved 11 2010, from <http://www.lionsmart.com/>
- [Mat08] Matsushita, T., Yabuta, K., Tsujikawa, T., Matsushima, T., Arakawa, M., & Kurita, K. (2008). **Construction of three-dimensional thermal simulation model of lithium-ion secondary battery.**, *Telecommunications Energy Conference, 2008. INTELEC 2008. IEEE 30th International*, 2008, 1 -6
- [Mer04] Merwe, R. , Wan, E., & Julier, S. (2004). **Sigma-Point Kalman Filters for Nonlinear Estimation and Sensor-Fusion: Applications to Integrated Navigation,** *AIAA Guidance, Navigation, and Control Conference and Exhibit*, 2004
- [Mi07] Mi, C., Li, B., Buck, D., & Ota, N. (2007). **Advanced Electro-Thermal Modeling of Lithium-Ion Battery System for Hybrid Electric Vehicle Applications.**, *Vehicle Power and Propulsion Conference, 2007. VPPC 2007. IEEE*, 2007, 107 -111
- [Ple04] Plett, G. (2004). **High-performance battery-pack power estimation using a dynamic cell model.** *Vehicular Technology, IEEE Transactions on* , 53 (5), 1586-1593.
- [Ple04b] Plett, G. L. (2004). **Extended Kalman filtering for battery management systems of LiPB-based HEV battery packs: Part 1. Background.** *Journal of Power Sources* , 134 (2), 252-261.
- [Ple04c] Plett, G. L. (2004). **Extended Kalman filtering for battery management systems of LiPB-based HEV battery packs: Part 2. Modeling and identification.** *Journal of Power Sources* , 134 (2), 262-276.
- [Ple04d] Plett, G. L. (2004). **Extended Kalman filtering for battery management systems of LiPB-based HEV battery packs: Part 3. State and parameter estimation.** *Journal of Power Sources* , 134 (2), 277-292.
- [Sim06] Simon, D. (2006). **Optimal State Estimation: Kalman, H Infinity, and Nonlinear Approaches** (1. Auflage ed.). Wiley & Sons.
- [Sti08] Stiftl, J. (2008). **Modellierung und Bewertung von Fahrzeugen mit seriellem Plug-In Hybridantrieb.** Hochschule Karlsruhe – Technik und Wirtschaft: Diplomarbeit.
- [Wie10] Wielgos, S., Brembeck, J., Otter, M., & Kennel, R. (2010). **Development of an Energy Management System for Electric Vehicles Design and System Simulation.** Technische Universität München.

Use of Modelica language to model an MV compensated electrical network and its protection equipment : comparison with EMTP

Olivier Chilard Jean-Philippe Tavella Olivier Devaux
EDF Research and Development
1, avenue du général de Gaulle, 92140 Clamart France
olivier.chilard@edf.fr jean-philippe.tavella@edf.fr olivier.devaux@edf.fr

Abstract

Today, neutral compensation based on Petersen coils is being applied in many MV electrical networks. The main reasons for this are :

- an important expansion of the underground network in rural environment. This leads to high phase-to-ground capacities of the outgoing feeders, which increases fault currents in networks using impedant grounding,
- an increased sensitivity of the customers to the quality of supply,
- changes in international standards (insulation coordination).

The introduction of arc suppression Petersen coils allows both to reduce the current in single phase-to-ground faults and to improve the quality of supply by reducing short supply disconnection.

Adapting this solution to the existing networks made it necessary to change the protection system, by using zero sequence wattmetric relays.

The feedbacks of the zero sequence wattmetric relays operating today show the need of an evolution of this protection equipment specifications. To address this need, EDF R&D has investigated the use of Modelica language for the electrical system modeling.

At first the relevance of Modelica language for electrical systems modeling has been studied. This work was made from a comparison of simulation results with those traditionally obtained with EMTP software, normally used by EDF R&D. This paper details this approach and underlines the interest of Modelica for the electrical network fields. The next step will be the use of the ModelicaML profile in order to establish a new version of the protection system specification.

1 Introduction

Currently, in order to design new control devices such as protection equipment, Matlab and EMTP are widely used. The solutions obtained are then validated on the field. Once a final solution is retained, a specification paper is then written and sent to the manufacturers which provide industrial solutions.

However, paper specifications are not formal, thus they may be interpreted differently by manufacturers and lead to some difficulties in conception phases. Moreover, when an equipment has been in operation for several years, if the initial solution has to be upgraded, the initial specification paper might be difficult to exploit again. Indeed, the informal description of the expected equipments often leads to a paper with redundant, missing or interpretable requirements.

In this context, a new approach based on the Modelica language has been studied. Modelica has been applied first in the particular case of the zero sequence wattmetric relay used in MV compensated networks. Moreover, this approach is intended to be used both for the development and specification phases.

This paper presents the modeling work and gives a comparison with a more traditional approach using EMTP-RV [3]. The simulation results are also provided and compared.

The final objective will be to provide a formal executable specification of the expected equipment.

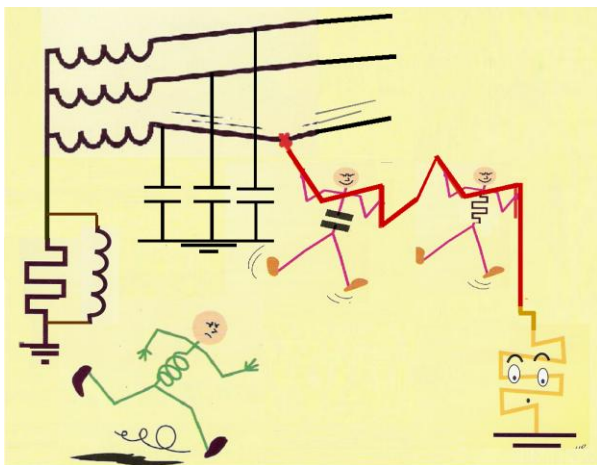
2 Description of the use case retained

The use case refers to the modeling and the simulation of two zero sequence wattmetric protections and a simplified MV compensated network. The protection is described in a specification paper established by EDF R&D. It has been developed to protect MV neutral compensated networks against mean and medium resistive single phase faults. This protection denoted PWH, is based on the analysis of both transient and steady state values of residual current and residual voltage available in the compensated MV substation.

Let's briefly recall the principle of a compensated network and that of the PWH.

In rural MV distribution networks, each feeder is mainly constituted of an overhead line and thus, when a fault occurs, the phase-to-ground capacitance current values are low (Figure 1). Therefore in these cases the neutral of the HV/MV transformer is grounded through a resistor and the fault current is equal to the neutral current.

Figure 1 : Principles of compensated networks

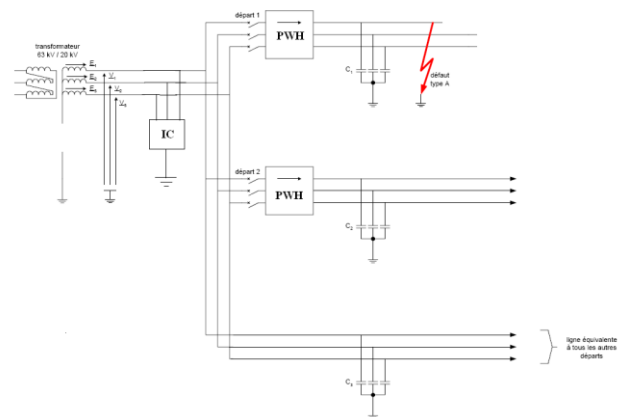


In mixed MV Distribution networks, an important part of each outgoing feeder is composed of underground cables. Therefore, contrary to the previous case, when a single fault occurs, the total capacitor current value seen at the MV busbar is important. Thus the resistor of the MV neutral grounding is replaced by a coil (a resistor in parallel with an inductor, refer to Figure 1) in order to compensate the zero sequence capacitor current and thus limit the current according to the international standard (insulation coordination). Consequently, if the mismatch of the coil is adjusted to zero Amps, the fault current is limited to the active neutral current part of the coil.

Concerning the network modeling, a simplified MV compensated network is considered. The following assumptions are made :

- The single phase fault is considered close to the MV busbar. Thus, the reactance of each outgoing feeder is neglected and only the phase-to-ground capacitance is considered,
- The MV substation three phase voltage source is assumed to be balanced,
- Consumption and reactive power compensators are not considered.

Figure 2 : MV compensated network retained



Therefore, the simplified MV network scheme retained is given in Figure 2. The substation is defined by its HV/MV transformer, its coil inserted on the neutral grounding and its MV busbar. One faulty feeder and two unfaulty feeders are considered. The second unfaulty feeder (without protection) represents an aggregation of all other unfaulty feeders. Two PWHs are also considered and respectively allocated to the faulty feeder and to the first unfaulty feeder (Figure 2). Its aim is to eliminate the single phase fault. To do that, the PWH has to send a trip to the circuit breaker of the faulty feeder.

A PWH is based on the analysis of both transient and steady state values of residual current and residual voltage measured at the MV substation.

PWH selective action is based on the known fact that the active component of the residual current in the faulty feeder is of the opposite direction and much greater than on any of the unfaulty feeders. Therefore, on the faulty feeder, the zero-sequence wattmetric relays detects negative zero sequence active power. On contrary, the protection placed on the unfaulty feeder detects a positive zero sequence active power.

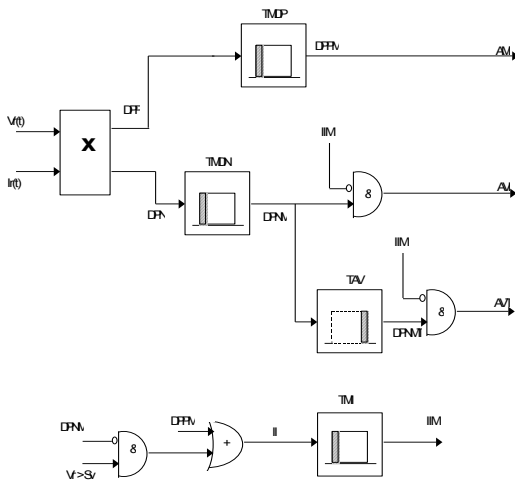
The PWH protection has to protect the network against two types of phase-to-ground faults :

- A permanent fault which involves residual current with a preponderant component at 50 Hz, after the transient phenomena due to the occurrence of the fault,
- A series of self-extinguishing faults, which is considered as the same fault.

In order to detect a permanent fault, the RMS values of the residual voltage & current and the residual active & reactive power are calculated for a cycle of 20 ms. For the two types of faults, residual power mean value is also calculated for a cycle of 60 ms (Figure 3).

Depending on the results of these calculations, according to a condition test, a positive or negative residual power message is send to the logical part of the protection which elaborates a trip protection message toward the circuit breaker of the faulty feeder.

Figure 3 : Principles of the PWH



3 Overview of the use case model implemented with EMTP-RV

3.1 Overview of EMTP-RV

EMTP-RV (ElectroMagnetic Transient Program) is a well-known software used for the simulation analysis of electrical power system.

The program is meant for solving problems such as :

- Switching transients and overvoltages,
- Short term analysis of disturbances,
- Overcurrent calculations,
- Control of electric drives,
- FACTS.

3.2 Use case modeling

3.2.1 EMTP Modeling approaches

Two types of components can be used under EMTP : traditional electrical components for electrical circuits and block diagrams for the description of automatic control of electrical systems. Most of the electrical components are not modifiable by users. Thus, if a new component is to be implemented, the user has the option of building it with block diagrams and elementary electrical components.

Another possibility is to compile a DLL or an S-function. The DLL approach consists of a causal description of the component according to the numeric solver of EMTP-RV. Thus, the DLL is a discretization of the mathematical representation of the component written in C language.

The S-function can be, for instance, provided by a model previously established under Simulink or Dymola tools. With this feature the numeric solver of the model is included in the S-function and can be different from EMTP's.

3.2.2 Use case modeling

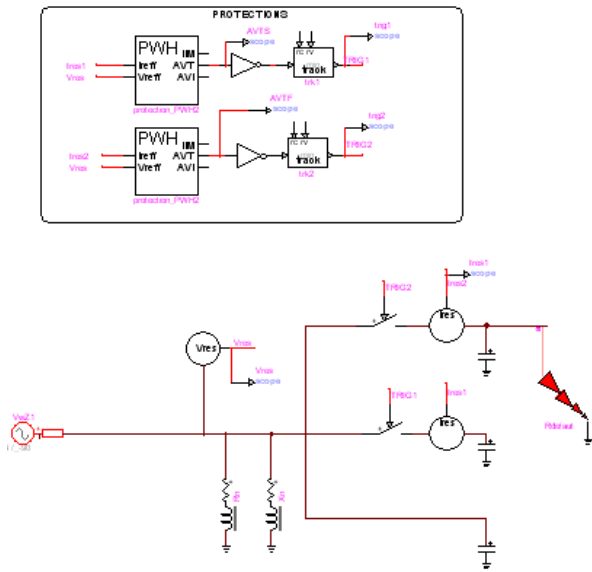
The MV compensated network model implementation is immediate. A predefined RL coupled impedance has been used in order to respectively define :

- The resistor and inductor (in parallel) of the neutral grounding impedance,
- The short circuit impedance of the three phase voltage source.

These components represent an equivalent model of both the transformer and the neutral impedance connected in its MV terminals (Figure 2). To do so, the positive, negative and zero sequence parameters are chosen for each element.

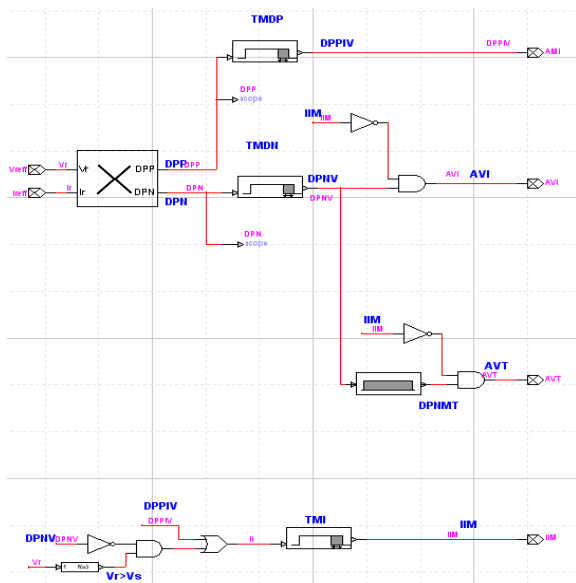
A simple capacitor grounding has been placed at each phase of each feeder.

Figure 4 : The use case implementation under EMTP



For the zero sequence wattmetric relays implementation a combination of different block diagrams has been used in order to model the PWHs. The resulting model, derived from the structure given in Figure 3 is shown in Figure 5.

Figure 5 : PWH model under EMTP

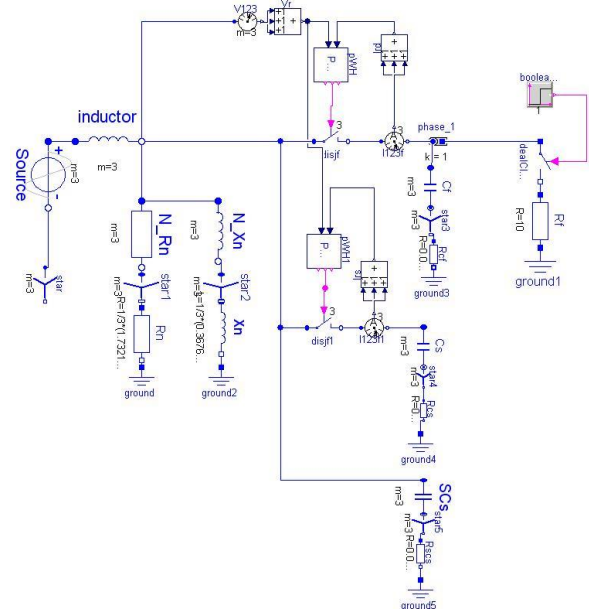


4 Overview of the use case model implemented with the Modelica language

Like under EMTP-RV, the simplified MV compensated network model implementation is immediate with Modelica. To do so, the components inside the official “Modelica.Electrical.Multiphase” package provided with the language have been used.

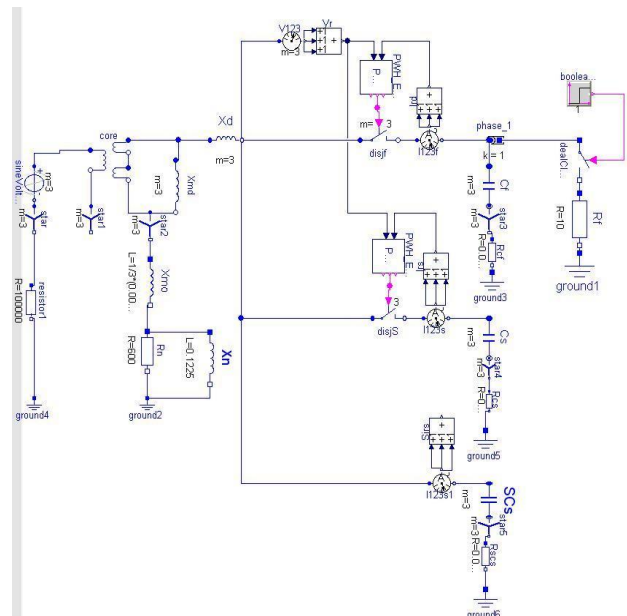
In order to simulate the use case, the Dymola tool based on the Modelica language has been chosen. It provides the numeric solvers required for the simulations and allows a graphical representation of the model.

Figure 6 : Use case implementation with Modelica



In order to do a first assessment of the possibilities provided by the “Modelica.Electrical.Machines” package, an alternative model with a transformer has been implemented too (Figure 7).

Figure 7 : Alternative modeling with a transformer



It is denoted that the RL coupled component is not available in the Modelica libraries. Thus, the R_n , X_n and X_{m0} given in Figure 6 and Figure 7 are defined as 1/3 of the difference between the zero sequence and positive sequence impedances expected with the positive sequence impedance placed in each phase (Figure 6 : N_{R_n} , N_{X_n} / Figure 7 : X_{md}).

In order to assess the Modelica language possibilities, the distribution MV Network has been also implemented without the use of the Modelica “Modelica.Electrical.Multiphase” package (Figure 8). In this case, coupled components have been created. To do so, the voltage and current variables of each connector are defined as a column vector with three elements belonging to each phase (Figure 9). Thus, each component has been defined by inheritance and from matrix equations (Figure 10).

Like EMTP-RV, zero sequence and positive sequence input parameters are given for each of these components.

Figure 8 : Use case implementation without the use of the “Modelica.Electrical.Multiphase” package

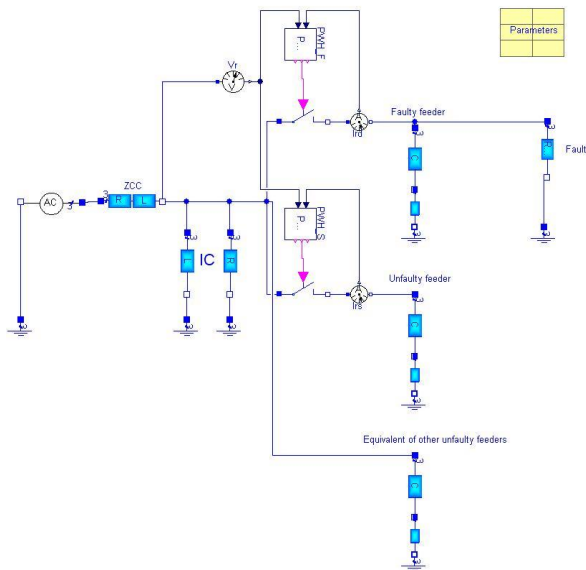


Figure 9 : Example of the ‘two pin’ partial model created without any standard component

```

partial model MIMOSA.electrical.Interfaces.Deux_bornes3
  "Deux bornes tri"
  Real v[3, 1];
  Real i[3, 1];
  plus3 plus;
  moins3 moins;
equation
  v = plus.v - moins.v;
  plus.i + moins.i = zeros(3, 1);
  i = plus.i;
end Deux_bornes3;

connector MIMOSA.electrical.Interfaces.plus3 "Borne plus tri"
  Real v[3,1] "effort";
  flow Real i[3, 1] "flux";
end plus3;
    
```

Figure 10 : Inductor L coupled model

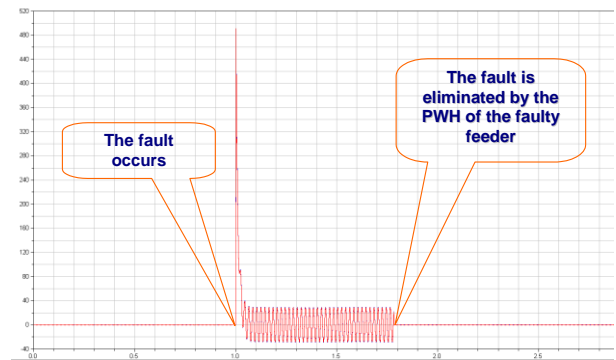
X_d and X_o respectively correspond to the positive and zero sequence impedances. The negative sequence impedance is taken equal to the positive one.

```

model Ldio "impédance L"
  extends MIMOSA.electrical.Interfaces.Deux_bornes3;
  parameter Real Xd "Ohms" a;
  parameter Real Xo "Ohms" a;
  Real Lm;
  Real Lp;
  Real L_matrix[ 3,3];
  parameter Real freq=50 "Hertz";
  constant Real PI=3.141592653589793;
  Real Ld=Xd/(2*PI*freq);
  Real Lo=Xo/(2*PI*freq);
equation
  Lm=(Lo-Ld)/3;
  Lp=Ld+Lm;
  L_matrix = [Lp, Lm, Lm; Lm, Lp, Lm; Lm, Lm, Lp];
  v = L_matrix*der(i);
end Ldio;
    
```

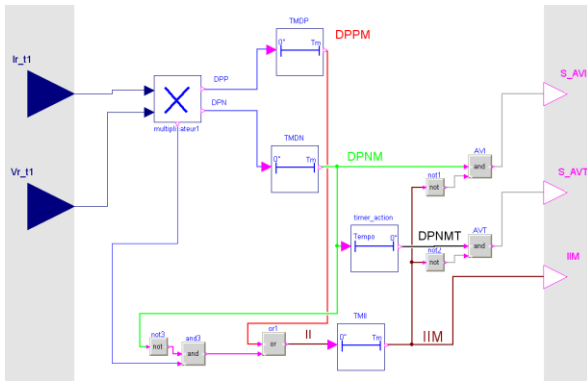
The three previously described Modelica models (with and without “Modelica.Electrical.Multiphase” package components) are equivalent as showed by the simulation results obtained in Figure 11.

Figure 11 : Fault current curves from the three models



For the zero sequence wattmetric relays implementation, a combination of the different block diagrams has been used in order to model the PWHs. However, these blocks have been created with the Modelica language in order to facilitate this work. The resulting model is presented in the Figure 12.

Figure 12 : PWH model under Modelica/Dymola



5 Modelica and EMTP

5.1 Validation of Modelica modeling

Some simulations have been done in order to validate the model developed with the Modelica language.

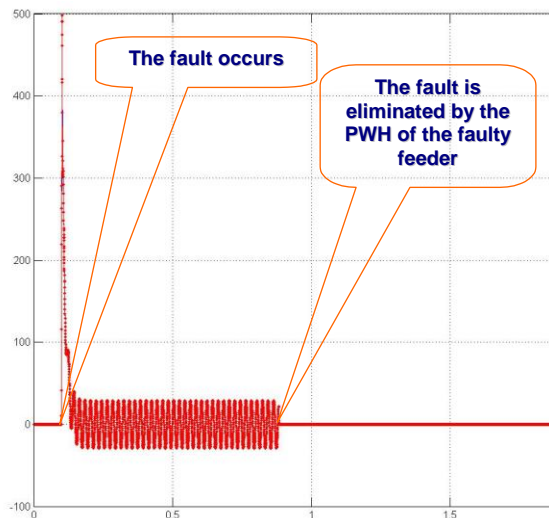
To do so, EMTP has been considered as the benchmark tool.

Under Dymola and EMTP, all the electrical signals belonging respectively to the PWH and the MV network are identical. In this paper, only the fault current is presented (Figure 13).

It is obtained from simulations performed on models given in Figure 4 and Figure 8. The fault current values are identical such as the fault clearance time due the PWH action.

Figure 13 : Phase-to-ground fault current

Blue : EMTP Red : Modelica/Dymola



5.2 Modelica and EMTP modeling comparison

The Modelica model, with all components created by the user (Figure 8) is compared with the EMTP model (Figure 4). EMTP has, for the electrical power part, implicit components and, for the control command part, elementary explicit blocks. These components are not modifiable and thereby to develop new components, the user has to :

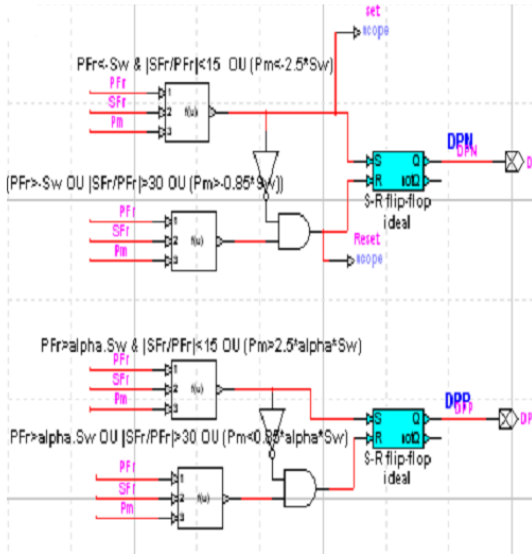
- Either create more or less difficult elementary components associations,
- Or build a DLL with S-function.

For the use case considered in this paper, only explicit and implicit blocks were used. Thus, given the basic components available in EMTP, some tricks have been used to model the treatment of the multiplier outputs named DPN and DPP (Figure 14). From this very simple case, it is not difficult to understand that it would be very difficult to implement and verify more complex systems such as distance protections or advanced functions developed in the SmartGrid. Although this may not be unfeasible (to be validated according to the available basic blocks), these developments would be difficult without any guarantee of success. Moreover, the level of readability of such models would be bad and this approach leads to an increase of the number of variables and thus unnecessarily constraints the solver.

In the opposite, Modelica models are simpler because the different blocks of the explicit PWH can be defined by the user. In other words, rather than associating basic blocks of the native library of Modelica, the user can create more adapted components or modify some existing blocks. Moreover, the Modelica language provides all specific language paradigms for algorithm developments and so, unlike EMTP, statements related to the treatment of the multiplier are close to the description given in the PWH specification paper (Figure 14 and Figure 15).

Figure 14 : Treatment of the PWH multiplier outputs (DPP & DPN)

Model under EMTP



Modelica Model

```

.
.
.
algorithm
// définition de la sortie DPN
if PFr < -Sw and abs(SFr/PFr) < 15 or Pm < -2.5*Sw then
  DPN := true;
elseif PFr > -Sw or abs(SFr/PFr) > 30 or Pm > -0.85*Sw then
  DPN := false;
end if;

// définition de la sortie DPP

if PFr > alpha*Sw and abs(SFr/PFr) < 15 or Pm > 2.5*alpha*Sw then
  DPP := true;
elseif PFr < alpha*Sw or abs(SFr/PFr) > 30 or Pm < 0.85*alpha*Sw then
  DPP := false;
end if;

//prise en compte de la valeur de la tension homopolaire

if VFreff > Sv then
  Vr_sup_seuil := true;
else
  Vr_sup_seuil := false;
end if;

end multiplicateur;

```

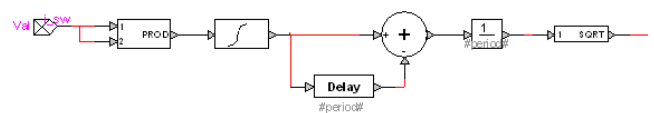
Figure 15 : PWH specification extract focused on the processing of the multiplier outputs DPN & DPP

- **DPN** (Détection Puissance Négative)
 $DPN = 1$ si $(PF_r < -Sw)$ et $(|SFr / PFr| < 15)$
 $DPN = 0$ si $(PF_r > -Sw)$ ou $(|SFr / PFr| > 30)$
- **DPP** (Détection Puissance Positive)
 $DPP = 1$ si $(PF_r > \alpha Sw)$ et $(|SFr / PFr| < 15)$
 $DPP = 0$ si $(PF_r < \alpha Sw)$ ou $(|SFr / PFr| > 30)$

Regarding the models related to the signal processing, for example the one in charge to calculate the RMS value of the busbar residual voltage, the description in EMTP and Modelica are similar. However, with EMTP, it is defined according to a connection of some basic blocks while it is mathematically declared in Modelica (Figure 16). By the other, it should be denoted that, although the type of the input/output blocks in the two cases are causal, with Modelica, unlike EMTP, the integral equation is acausal. It is a very important feature of Modelica that allows the user to declare a system without worrying about the order of the equations.

Figure 16 : Model providing the RMS value of the busbar residual volatage

Model under EMTP



Modelica Model

```

model multiplicateur "input Vr et Ir output DPN et de DPP"
.
.
parameter Real window=0.02
.
Real Y1;
.
Real VFreff;
.
equation
.
VFreff = sqrt(abs((1/window)*(Y1 - delay(Y1, window))));
der(Y1) = Vr_t*Vr_t;
.
.

```

Moreover, contrary to EMTP, it is important to underline that all Modelica components are not solver dependent. Under EMTP, both with a source code or a DLL, a C code program of the discretized equations of the system should be created according to the EMTP solver. In the opposite, this task is automatically performed by the symbolic solver in the Dymola tool.

6 Conclusions

It is undeniable that the opening provided by Modelica allows an unmatched flexibility in modeling. Contrary to EMTP, the PWH model is very close to the description given in the paper specification. In other words, Modelica description corresponds to a formal version of the specification while under EMTP, given the basic blocks available, the model obtained is rather a translation of the specification.

In addition our study shows the great interest of the Modelica language for power electrical system modeling and for electrotechnical studies.

So, for EDF R&D, the following steps will be :

- To use actual measurements as input to the PWH Modelica model in order to study more precisely the behavior of the protection,
- If required, to update the model and create a new release of the specification including the textual Modelica model.

For this work, one of the intended goals will be to examine the ModelicaML profile [2] as an opportunity to graphically specify a system behavior.

References :

- [1] Spécification Technique EDF
HN 45-S-54
Régime de neutre compensé -
Spécification de la Protection Watt-
métrique Homopolaire
- [2] <http://www.openmodelica.org/index.php/developer/tools/134>
ModelicaML - a UML profile for
Modelica
- [3] <http://www.emtp.com/>
EMTP-RV - Computational Engine
- [4] Principles of Object-Oriented
Modeling and Simulation with
Modelica 2.1
Peter Fritzson

The Vehicle Dynamics Library: New Concepts and New Fields of Application

Johan Andreasson
 Modelon AB
 Ideon Science Park
 SE-22370 LUND
 johan.andreasson@modelon.com

Abstract

The Vehicle Dynamics Library is a commercial Modelica library for vehicle dynamics applications. This paper highlights recent development with focus on extended usability. Key changes are improved interoperability with other tools, improved simulation performance, extended vehicle system simulation, and expanded analysis. Examples are given from efficient simulation of drivelines, development of active safety systems, and quasi-steady-state analysis, among others.

Keywords: Vehicle dynamics; mechanics, active systems, quasi-steady-state analysis

1 Introduction

The Vehicle Dynamics Library (VDL) [2] is a commercial Modelica library providing a foundation for model-based vehicle dynamics analysis. Since the introduction of the library in 2006, there have been significant extensions and improvements. In this paper, some of these and their fields of application are discussed.

The scope of VDL spans from classic vehicle mechanics analysis to full vehicle system simulation and evaluation. One of the fundamental guiding principles of the library is the ability to mix between behavioral and physical models to make it possible to conveniently change, not just between different configurations, but also between different levels of detail.

As such, VDL is designed with several aspects in mind and in this paper, the contents is focused on four main aspects; Section 2 focuses on the work on the mechanical models of the vehicle while Section 3 treats the system aspects of the vehicle, in this case meaning the part of the vehicle that is used to control the vehicle mechanics. Section 4 describes further

options to interface VDL with other tools and Section 5 describes extensions to the scope of analysis.

As this paper presents the incremental work relative to what is presented in [2], it is strongly recommended referring to that work for a more thorough background to VDL and its architecture.

2 Vehicle Mechanics

Here, the focus is on the work driven by improvements in configurability and simulation speed. The two first sub-sections explain the improvements of the suspension models while the latter focus on mechanics related to driveline and brakes.

2.1 Extended Suspension Templates

Typically, a suspension, just like any part of the vehicle is configured from different templates. This allows to conveniently exchanging joints and parts while maintaining the topology. Main reasons for this are to improve working efficiency and reduce model maintenance [3].

To facilitate the configurability also of the suspension topology, new components have been introduced that allow also for topology changes to the connection structure to be made based on parameter settings. As illustrated in Figure 1, the stabilizer mount (a) can be connected to the upper A-arm (b), the upright (c) or the lower A-arm (d). In this example also the spring mounts connections can be reconfigured which in this case covers 18 different topology configurations. Further customization is straightforward for the user if necessary.

Unlike some other multi-body simulation software, the Modelica formulation is transformed so it can be integrated by standard index 1 solvers. This method has several advantages and is one of the keys to the inherent multi-engineering capabilities. One implication is that a closed mechanical loop gene-

rates implicit constraint equations that have to be solved, either symbolically or numerically.

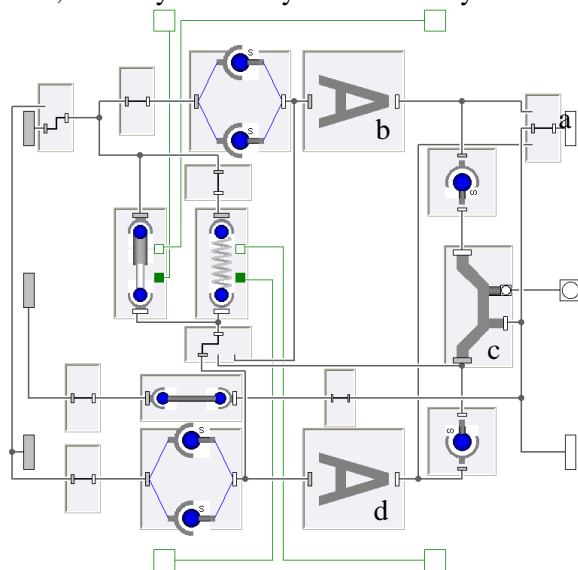


Figure 1 a reconfigurable suspension linkage template where each element can be replaced.

To support symbolic solutions, which normally is the much faster alternative, a dedicated formulation was introduced in `Modelica.Mechanics.MultiBody` [9,1] and is based on the idea that for a sequence of joints that in total have six degrees of freedom, no constraints are required if a composite model representing all the joints is created.

This type of formulation can significantly improve simulation speed as the lack of constraint equations also eliminates the corresponding nonlinear systems of equations in the resulting simulation code. This is important especially for real time simulation. In VDL, the concept has been modified slightly to allow for users to replace individual linkage components within the composite model.

The concept is illustrated in Figure 2, showing the same suspension topology as seen in Figure 1, with the difference that the upper control arm (a), the king-pin (b) and the tie rod (c) is represented as one composite joint without any constraints. This mechanism defines the wheel carrier motion (d) given the motion of the chassis (e), the lower control arm (g) and the steering rack (f).

From a user perspective, there is no difference between the models in Figure 1 and Figure 2, except for the improved simulation speed; all other VDL features such as force visualization remain the same.

2.2 Behavioral Suspensions Models

Behavioral suspension models are a common way to represent suspension characteristics in a convenient way. The idea is to record how the wheel carrier

moves depending on the degrees of freedom in the suspension, and to capture that in functional representations, e.g. by tables or polynomials.

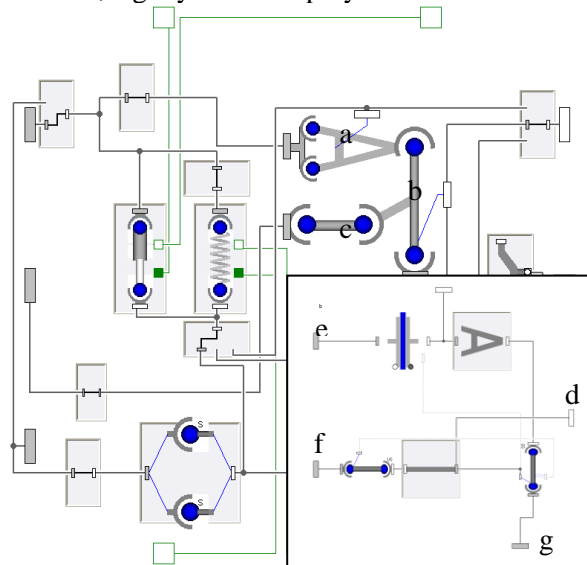


Figure 2 the same suspension topology as in Figure 1, but with an efficient model formulation to improve simulation performance

In VDL, the focus has been on supplying a complete functional representation so that the result should be identical with any kinematically well defined suspension. This includes not just the actual characteristics, but also the transmitted reaction forces and torques. This is essential especially for steering design.

The suspension kinematics is represented by a set of tables, where the number of dimension depends on the degrees of freedom of the linkage. For an independent front suspension linkage, the representation uses two dimensions, one for suspension travel, and one for steering, giving eight functions, three for hub position and orientation, and one for spring and damper compression, respectively.

The tabular representation has the same interface as a multi-body variant. This modular approach allows selecting parts of the chassis or suspension to be implemented using tabular characteristics, and other parts to be represented with traditional multi-body implementations.

To address the compliance which typically is present in a suspension, due to elastic bushing elements or material compliance, the effects of the compliance is in the behavioral case super-positioned on the kinematic motion. This can be done, either separately for each linkage, or lumped for the whole suspension.

The suspension compliance, just as the kinematics, can be calculated from a more complex model, but a common scenario is to get compliance and/or kinematic information from measurements on real

vehicles. Typically, the compliance information is given as compliance matrix \mathbf{C} (inverted stiffness) rather than stiffness as typically is used in simulation software. For a model described as a spring-mass system

$$F = \mathbf{K}\Delta, F = \mathbf{M}\ddot{\Delta},$$

the compliance data renders two problems: First, $\mathbf{K} = \mathbf{C}^{-1}$ has to be generated but \mathbf{C} is not necessarily full rank and probably ill-conditioned. The reasons may be many, for cost and time reasons for example, often only parts of \mathbf{C} are measured. As a result, manual work and assumptions are often required to compute \mathbf{K} . Second, the relation between \mathbf{M} and the resulting \mathbf{K} are often such that the Eigen-frequency is much higher than the frequency range you are considering when using this type of models. Simulating this would lead to a significant performance loss.

The latter problem described above is in this context actually an opportunity to reformulate the model. Since the inherent frequency of the system is unwanted, the compliance data can be used directly in a model that is formulated as

$$\Delta = G(s)\mathbf{C}F,$$

with $G(s)$ being a second order system with a defined cut-off frequency. This frequency can be set by the user so that the model's static behavior is identical to the spring-mass system, and with a dynamic response that is fast enough for the performed analysis.

A further advantage with this approach is that the arbitrary values in \mathbf{C} can be set to zero, corresponding to the removal of degrees-of-freedom, without the need to change the model topology. It is therefore easy to switch between a compliant and a rigid version of a suspension, by just modifying the parameters. With the spring-mass approach, such a change requires a recompilation of the model before it can be simulated. The approach has been used in e.g. [8] and is also suitable to use with kinematic models for real time simulation purposes.

2.3 Driveline and brake mechanics

In [2,3] Rotational3D was introduced, a concept that allows for the three-dimensional effects of 1D rotational mechanics to be captured with a straightforward representation. Compared to the MultiBody approach, simulation performance improvement is in a typical case around a factor 20, the details are explained in [4].

Since then, focus has been on reducing computational cost in drivelines and brakes further by improving the performance of the hybrid elements, especially friction. With the reached performance increase, one can conveniently model and simulate

different active drivelines without any considerable simulation slowdown. The new friction components are provided with the Modelon.Mechanics library and these are the recommended choice for vehicle subsystems such as brakes and powertrains, especially in real-time critical applications.

The driveline topology has been reworked to facilitate the configuration of e.g. user defined differentials. The idea is to use Rotational3D to build housings for e.g. the differential, and then to use components based on the standard Rotational formulation. This is illustrated for two differentials in Figure 4; the open differential to the left consists of three Rotational3D components corresponding to the bearing of each axle (a), and two components for visualization (b). The other components; the cut component (c), the pinion-ring gear (d) and the differential (e) are purely one-dimensional components.

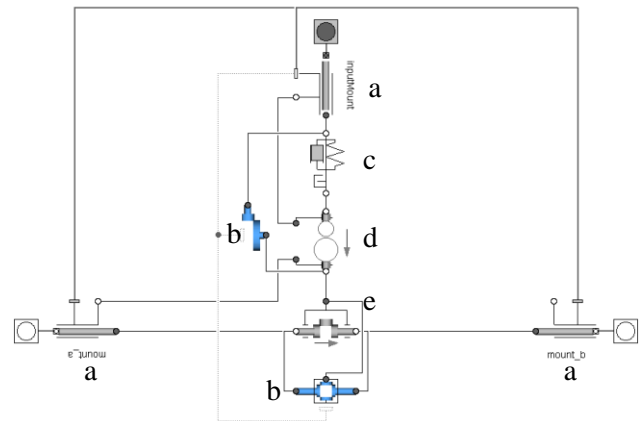


Figure 3. The layout of an open differential using standard Rotational components in a Rotational3D housing. Components with blue icons represent graphical information only.

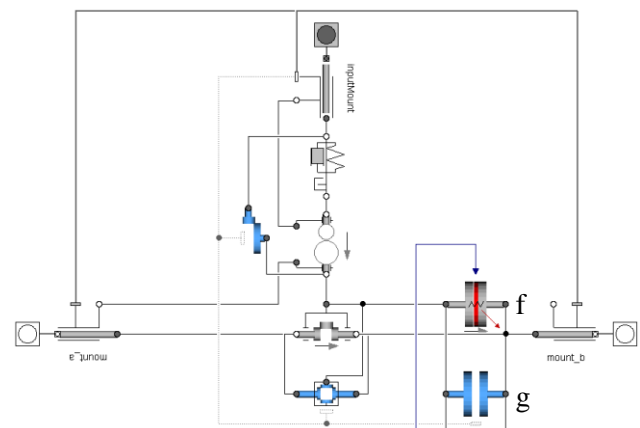


Figure 4. The differential from Figure 3, extended with a clutch between the differential case and the right axle, used for slip control.

One reason to separate the differential from the ring-pinion gear is seen in Figure 4; the clutch controlled differential is an extension with an additional 1D clutch (f) and a component for its visual proper-

ties (g). The clutch is connected to the differential case and the right axle, allowing the differential to lock.

To facilitate the analysis of the performance of drivelines, Rotational3D has been extended with built-in visualization of torque flow. The direction of the torque flow visualization does not visualize the sign of the torque but the direction of the power flow. This gives a quite intuitive interpretation of the visualization of the results. An example is given in Figure 5.

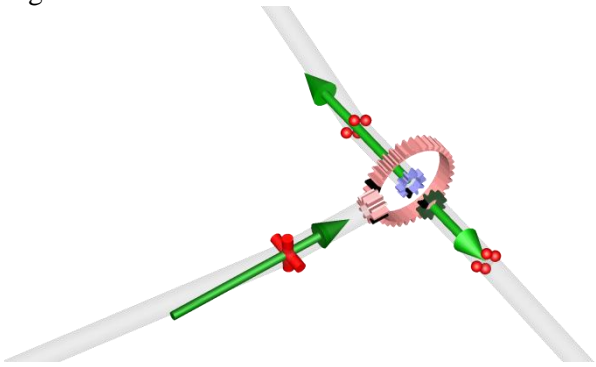


Figure 5 Visualization of a driveline where more torque is distributed to the right (top) shaft than to the left (bottom)

3 Vehicle Systems

The pure mechanical part of vehicle dynamics does and will continue to decrease as different active systems play an increasing roll for vehicle behavior. VDL was designed with this in mind from the start and recent improvements have been focused on extending this functionality. One significant part of this work is to include behavioral models that make it easy to quickly get to a minimal representation of the complete system, and then from there be able to select what details to focus on. There is a clear analogy to the tabular suspension models in Section 2.1 that are used to reduce complexity of the chassis. Here, it is illustrated for two active safety systems on one hand, and electrical or partly electrical propulsion on the other.

3.1 Electronic Stability Control

Functional representations on common safety systems have been implemented, including the required actuators and sensors, as well as sample architectures. A first example is an embedded brake system controller with anti-lock braking, yaw stabilization, brake force distribution and traction control, as seen in Figure 6. The signal bus is defined as an inner/outer expandable connector, which is similar to a globally available namespace. This allows for con-

trollers, sensors and actuators to be anywhere in the model hierarchy.

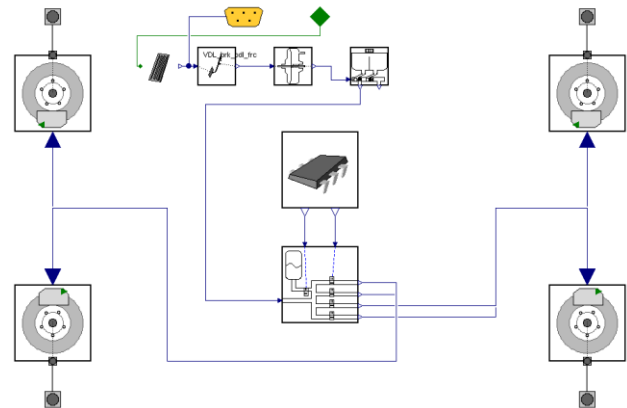


Figure 6 Brake system with modulator, and embedded brake system controller.

3.2 Brake Assist

The models can easily be copied and modified by the user to fit their specific needs. An example of a brake assist controller illustrates this. It is based on the previous example, and extended with a centralized vehicle controller that incorporates information from a distance sensor to add brake action when approaching the object ahead either too fast and/or too close.

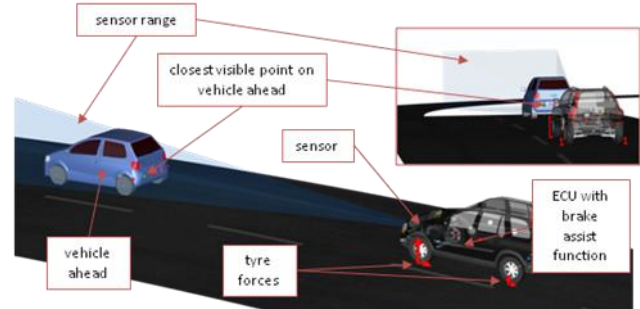


Figure 7 Animation view of a test scenario for a vehicle equipped with brake assist.

The modular approach makes it easy to add, extend or modify the existing architecture or to build a new one from scratch, whatever fits the application better. In any case, a user can select what part of the safety system should be modified, a typical scenario is to use the ABS and ESC model when doing initial studies and switch to more detailed models for verification purposes.

3.3 Electrical actuation

Modelica is of course a natural platform for multi-disciplinary investigations with VDL and libraries like Smart Electric Drives (SED) [11] or SPOT [12], detailed models of vehicles with electrical actuators can conveniently be modeled [7]. In many cases,

however, the need to start on a more conceptual level is fundamental.

To facilitate conceptual analysis, behavioral models for electric systems are introduced. The idea is to parameterize these based on a minimal set of output-related properties such as power, torque and efficiency for machines and then use this information to calculate the electric power consumption.

The models are defined with the same interface, mechanical and electrical connectors, to allow them to be replaced with more detailed versions whenever needed.

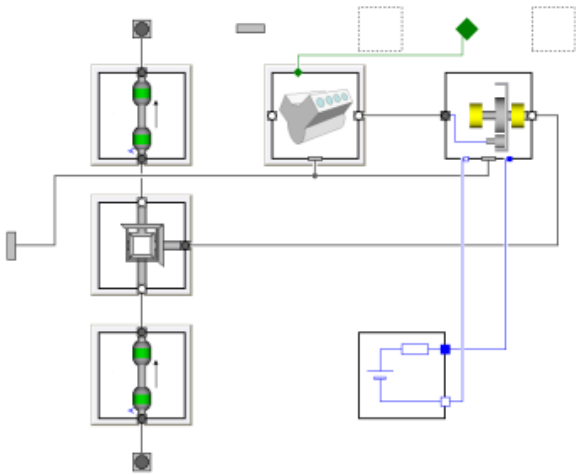


Figure 8. Hybrid electric driveline where components can be exchanged. This allows for convenient switching between detailed and behavioral models.

4 Interfacing

With the use of Dymolas source code and binary export capabilities, the use of VDL spreads to various applications such as in customized track-side tools and as vehicle models in driving simulators and other vehicle dynamics simulation software [13]. This section elaborates on some key improvements in VDL to facilitate such applications.

4.1 External Ground Representations

VDL has a ground representation that is based on a herring-bone representation, described as depending on two independent coordinates. This allows for an efficient representation with high resolution where needed. The interface allows for various implementations, both in Modelica and as external code. By supplying information about the position, heading direction and normal based on these coordinates, VDL is able to calculate contact points.

Recently, this has been extended to allow for user to supply own routines for contact point calculation, this in turn facilitates the use of external ground representations further and allow for more convenient plug-in of VDL models in other types of environments, such as driving or traffic simulators.

For full functionality, three different contact point calculation methods must be supplied as described below and illustrated in Figure 9 and Figure 10.

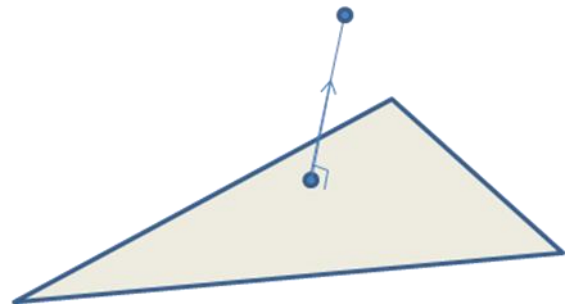


Figure 9 Closest point on surface.

The closest point on the surface, Figure 9, is the default contact point, defined so that the third ground coordinate corresponds to the height over ground along the road normal. This representation is used for e.g. ground impact models.

For a wheel, the first representation is however not a suitable representation if the wheel is inclined relative to the surface. This as the resulting point would diverge from the actual tyre contact. In this case the closest point on the surface that lies within the plane of the wheel should be returned, Figure 10, left.

The third method returns the intersection between the ground and a line defined from the point along a predefined vector a relevant representation, Figure 10, right.

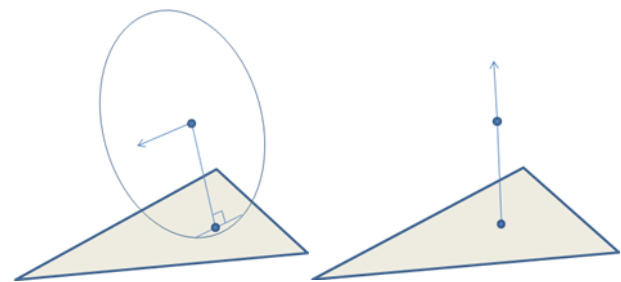


Figure 10 Closest point on surface that lays within a plane, left, and intersection point between line and surface, right.

4.2 Ground Generation

For the tabular ground representation, the RoadBuilder was introduced in [1]. It conveniently generates road data from input like curvature and banking. The functionality of the RoadBuilder has been

extended to also handle trajectories defined by measured points, and closed loop circuits.

4.3 Ground Access

Information about the ground can always be accessed from within Modelica by direct function calls. To facilitate usage and to make sure that the most efficient calculations are used, a block set has been developed that allow for users to conveniently build own models that requires ground information.

In Figure 11 this approach is being used to allow for simple tire models to be used on non-smooth surfaces. It works as follows: The original contact point (a) is used as a reference to create a compression profile of the tyre (b), and by a weighting of this information, the resulting contact point (c) can be calculated. With this representation underlying surface model can be made to work with standard single contact point models.

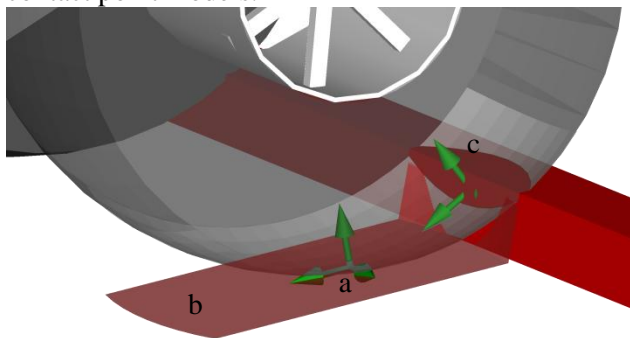


Figure 11 application of contact point filtering for a wheel traversing a cleat.

4.4 External Tire Models

The reconfigurability of VDL makes it easy to include external tire models. This can be done in several ways. The two most common ways are to either replace the VDL tire force calculation only while maintaining contact point calculation in VDL, or to replace the wheel including the contact point calculation. The first method is commonly used to incorporate in-house tire models while the latter is used to interface e.g. DelftTyre [6] and FTire [5].

5 Analysis

Since earlier, VDL provides a various experiments for dynamic simulation for full vehicles and subsystems. This section especially focuses on how also steady-state, and quasi steady-state analyses can be performed. Also, an expansion of the capabilities of the sensor suite is motivated.

5.1 Sensors

For the analysis of vehicle dynamics, sensors are required that can generate forces with respect to different coordinate systems. A typical example is the lateral acceleration of the car that with respect to the car body corresponds to what you would measure on a test drive, and with respect to the ground or the trajectory that corresponds to the cornering capabilities of the car.

To be able to conveniently change between different relevant coordinate systems, the standard set of the sensors has been extended with further options to resolve output. Especially, the vehicle frame projected onto the ground surface is added to support cases as described above. This is defined either according to the ground plane under the frame, or as a plane defined by averaging the wheel contact points. These planes coincide with the world horizon for flat roads.

5.2 Steady state and quasi steady state analysis

In many analyses, the dynamic response of the vehicle is not of interest; instead the focus is on the steady-state characteristics. Particularly in racing this type of analysis is a central part in balancing and tuning of vehicles. In Modelica, the complete experiment is defined as

$$F(p, u, x, \dot{x}) = 0,$$

with p , u , x , and \dot{x} are parameters, input, states and state derivatives, respectively. For dynamic simulation, this is transformed into a formulation that a standard integrator can handle, i.e.

$$\dot{x} = f(p, u, x).$$

Finding the steady state solution in many cases means solving this equation for $\dot{x} = k$, where k often but not always are zeros. The unknowns are typically combinations of (u, x) . When approaching the problem from a solver point-of-view, the result is a non-linear system of equations that in many cases is difficult to handle. There is ongoing work with Homotopy methods to improve the capabilities of the tools in this respect [10].

Practically, there are some further aspects to consider, and the most important one is that you want an answer even if there is no mathematical solution. A typical application is to take the car to its limit in some sense. The purpose of the analysis is to figure out where that limit is so any approach will at some point shoot over the target.

VDL has been extended to handle this in a robust way. The idea is to modify the model so that the steady state formulation is presented to the solver as a dynamic simulation. The states that describes the

principal motion of the car, x_p , is separated from the other states and u_p is introduced for the corresponding inputs. A set of residual flow variables, R_p , is introduced to match the principal states and then u_p is used to minimize R_p . To the tool, this is presented as

$$G(p, (u', x_p), (x', u_p), (\dot{x}', \dot{u}_p)) = 0,$$

where principal states now are inputs and principal inputs now are states. The structure of this formulation makes it straight-forward for the tool to transform it into

$$(\dot{x}', \dot{u}_p) = g(p, (u', x_p), (x', u_p)),$$

which can be solved with standard integrators. From this simulation one can gain knowledge of all the resulting vehicle states as well as the required driver input. At any point in time, R_p , will give a measure on the validity of the solution. Figure 12 shows the result from a quasi steady-state analysis. The car is set-up at a defined point along the race track, in this case a corner. The speed is increased while cornering curvature is maintained until the lateral acceleration capabilities are exceeded. The screen shots shows the car at different stages of this test.

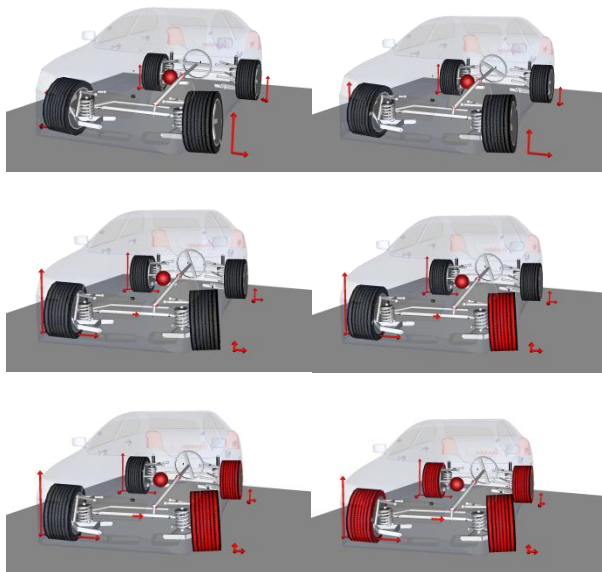


Figure 12 Steady state corner exit solutions for 10, 50, 90, 95, 98, and 100 percent of the car's capacity. Red wheel color indicates saturation. Arrows indicate tire forces. Note the change in steering wheel angle.

Summary

This paper highlights new features in VDL with emphasis on improved interoperability with other tools, improved simulation performance, and an expanded range of analysis.

References

- [1] Andreasson J et al, Modeling of a racing car with Modelicas MultiBody library. In: Proceedings of the Modelica 2000 workshop, Lund, Sweden, October 2000
- [2] Andreasson J et al, The VehicleDynamics Library - Overview and Applications. In: Proceedings of the 5th Modelica Conference, Vienna, Austria, 4-5 September 2006
- [3] Andreasson J, On Generic Road Vehicle Modelling and Control, PhD thesis, ISBN 91-7178-527-2, 2006
- [4] Andreasson J et al, Rotational3D - Efficient modelling of 3D effects in rotational mechanics. In: Proceedings of the 6th Modelica Conference, Bielefeld, Germany, 3-4 March 2008
- [5] Beuter V, An Interface to the FTire Tire Model. In Proceedings of the 8th Modelica Conference, Dresden, Germany, 21-22 March 2011.
- [6] Drenth E et al, Modelica Delft Tyre Interface. In Proceedings of the 8th Modelica Conference, Dresden, Germany, 21-22 March 2011.
- [7] Gerl J et al. Multi-Domain Vehicle Dynamics Simulation. In Proceedings of the 8th Modelica Conference, Dresden, Germany, 21-22 March 2011.
- [8] Jonasson M et al, Modelling and parameterisation of a vehicle for validity under limit handling, In: Proceedings of the 9th International Symposium on Advanced Vehicle Control (AVEC'08), Vol. 1, pp. 202–207, Kobe, Japan, 2008.
- [9] Otter M et al, The New Modelica MultiBody Library, In: Proceedings of the 3rd International Modelica Conference, Linköping, Sweden, November 2003
- [10] Sielemann M et al. Robust Initialization of Differential-Algebraic Equations Using Homotopy. In Proceedings of the 8th Modelica Conference, Dresden, Germany, 21-22 March 2011.
- [11] Smart Electric Drives, <https://www.modelica.org/libraries/SmartElectricDrives>
- [12] SPOT, <https://www.modelica.org/libraries/spot>
- [13] Ziegler S et al, Extending the IPG CarMaker by FMI Compliant Units. In Proceedings of the 8th Modelica Conference, Dresden, Germany, 21-22 March 2011.

Modeling and Simulation of Gear Pumps based on Modelica/MWorks®

Chen Liping¹ Zhao Yan¹ Zhou Fanli¹ Zhao Jianjun¹ Tian Xianzhao²
¹CAD Center, Huazhong Univ. of Sci. & Tech., Wuhan, China, 430074
¹Suzhou Tongyuan Software & Control Tech. Co., Suzhou, China, 215123
 chenlp@hustcad.com zhaoyan808@foxmail.com
 {fanli.zhou,jjzhao168}@gmail.com, tianxz@tongyuan.cc

Abstract

In this paper, we present a new method of modeling the hybrid external gear pumps based on Modelica and assess it experimentally. We model the whole working process of an external gear pump. The chamber of the pump is divided into a set of Control Volumes (CVs), whose effective volumes change along with the rotation of the gears. The CVs take in fluid from the inlet port and squeeze fluid out at the outlet port. The whole model of the pump also takes into account flow ripple, pressure distribution, leakages, meshing conditions, etc. Details of each component of the whole pump are provided. From the pressure distribution in space of the gear tooth, we can calculate radial force on shaft, based on which the shaft motion can be simulated.

We carry out a set of predictions in MWorks and report some results on the post processing. These results are consistent with those from the experimental data.

Keywords: Hybrid Gear-Pump Modeling; Control Volumes; Pressure Distribution; Radial Force on shaft; Modelica

Nomenclature

B the gear thickness
 h_{c1} the radial clearance of the bearing
 h_{c2} the axial clearance of the bearing
 n the tooth number

1 Introduction

The gear pumps are among the oldest and most commonly used pumps in the industry. It has become the main choice for fuel system designers due to long life, minimum maintenance, high reliability, capabil-

ity to operate with low lubricating fuel, low heat input to fuel, small size, and low weight.

External gear pumps use a simple mechanism (two gears) to generate flow and therefore have a minimum number of parts associated with the design. However, many factors ignored in the design, for example the volumetric efficiency and the pressure peaks, may greatly influence the performance of the pump.

In order to improve the design of the pumps, a good first step is to develop a mathematic model for the simulation. Some work focusing on different aspects of this aim has been done in the past few years.

Manring and Kasaragadda [1] presented an approach to finding a solution for the instantaneous length of action for the two contacting teeth in order to study the flow ripple. In theoretical research, pumps are considered in an ideal case: (1) the fluid is incompressible, (2) fluid leakage is neglected, and (3) the pump parts are inflexible.

Heisler et al [2] used the computational fluid dynamics (CFD) to better analyze the effect of modifying the design of an existing external gear pump. This paper also developed a new approach to simulating the helical gears. Although the computational time and model complexity have been reduced, simulations can be done only under some restrictive assumptions, including restricted flow domain, definite values of the boundary conditions, and fixed position of the gear axes.

Wahab [3] presented a simple analytical and explicit approach to predicting the leakage flow rate under different inlet/outlet pressure differences.

Castilla et al [4, 5] studied experimentally the movement of the shaft of a driven gear in a gear pump, in particular the dynamics of the shaft in the journal bearing of a gear pump.

Elia [6] developed a mathematical model that simulates the running in of external gear pumps, taking into account the factors mentioned above. Be-

sides, the model also estimates the quantity of material taken away.

Following a different way, Vacca et al [7] provided a global description of the pump or motor. They implemented a model, named HYGESim, in the AMESim® platform. The model uses both standard sub-models and C++ sub-models developed by the authors.

Though valuable in some aspects, these studies are unable to develop a global hybrid model, which is the goal of our paper. We construct a hybrid model of the external gear pumps by using a multi-domain modeling language, Modelica on MWorks® [8]. The model takes into account the leakages, compressibility of the oil, flow ripple, pressure distribution, etc. It can also calculate forces and torques, which are the base of modeling the shaft motion.

The paper is outlined as follows. Section 2 describes the mechanical system. Section 3 gives an overview of the theory on which the pumps are modeled based and details about each component of the whole pump. Section 4 reports the simulation results and compares them with those in [7]. Section 5 concludes the paper.

2 Pump Description

Figure 1 shows a cross-sectional view taken through the gears of a typical gear pump.

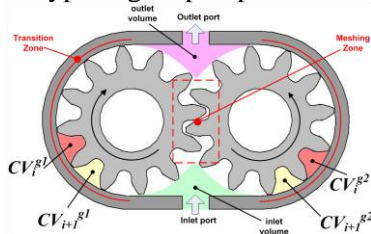


Figure 1 Control Volumes Defined in the Fluid Dynamic Model

Note: like most actual gear pumps, this pump has two identical gears to displace fluid. The superscripts $g1$ and $g2$ denote the driving and driven gears respectively. The tooth number of gear 1 is the same as that of gear 2. The **Control Volumes (CVs)** within the tooth gaps are variables dependent on the gear angle. The subscripts i is the index of these Control Volumes. The Outlet and Inlet volumes are two fixed volumes at the outlet and inlet ports respectively.

To produce a flow within a gear pump, fluid is carried by the CVs from the intake side of the pump to the discharge side of the pump through the transition zone. As the gears rotate, these CVs increase their pressure to when reach the high-pressure chamber. As the gear teeth mesh in the meshing zone, fluid is squeezed out of each tooth gap by the mating

tooth. When two tooth pairs contact, a trapped volume is generated. This may cause positive pressure peaks and the onset of cavitation. (To avoid this, the trapped volume must be connected to the high or low pressure chambers, which is the role of the relief grooves in the lateral bushes.) On the intake side, the gear teeth are coming out of the mesh. The volumes of CVs increase so that fluid is inhaled into the tooth gaps. This process repeats itself for each revolution of the pump and therefore displaces fluid at a rate proportional to the pump speed.

3 Model Description

3.1 Overview

In this section, we first give an overview of the pump model. Some components come from a free library HyLibLight [9] based on Modelica.

The aim of this paper is to construct a hybrid model of the external gear pumps. It considers the leakages, compressibility of the oil, flow ripple, pressure distribution, etc. Specifically, the pump is modeled under the following assumptions:

- 1) The gears and the housing case are rigid; only the oil is compressible.
- 2) The position of the shaft is known before the simulation and is fixed during the simulation.
- 3) The pressure at every single isolated region and fixed volume is well-proportioned.
- 4) The temperatures in all CVs are the same and constant.
- 5) The tooth numbers of the two gears are the same.

In **Figure 1**, a pump is divided into $2n+2$ control volumes, where n is the tooth number of a gear. **Figure 2** shows the flows between those control volumes. This method is similar to but not the same as that in [6]. There is no variation in the number of the control volumes which causes variation in the number of differential equations.

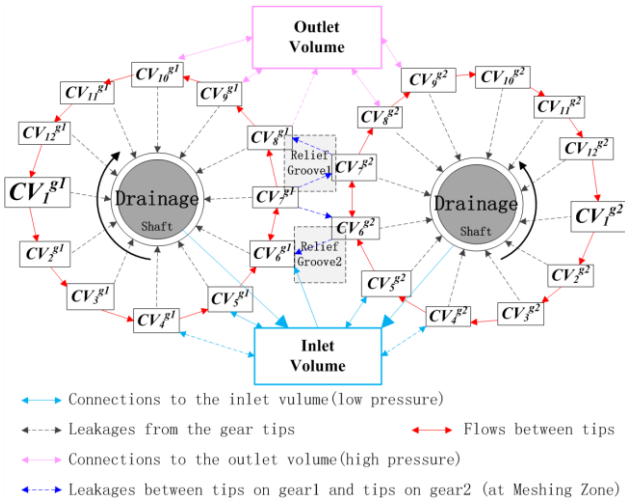


Figure 2 The Flows Between the Control Volumes

Note: arrows do not mean the flow directions; the direction of the flow depends on the pressure differential and the rotation direction. As the gears rotating, the channels of some flows are shutting down and some of others (not shown in this figure) are turning on.

Dark blue lines in the meshing zone represent the leakages from a trapped volume on one gear to a CV on the other gear.

The flow between CV_i^{g1} and CV_{i+1}^{g2} is the leakage through the clearance between two meshing surfaces;

The flow between CV_i^{g1} and CV_i^{g2} is the leakage through the clearance between two un-meshing surfaces.

Relief Grooves are chambers to which the CVs connect in turn. Flows from the CVs to the Relief Grooves in the meshing zone are shown in **Figure 3**.

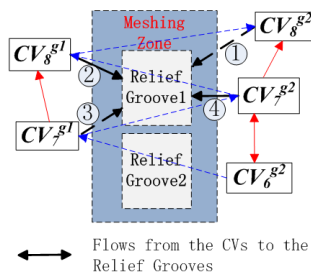


Figure 3 Flows from the CVs to the Relief Grooves (RG)

RG1 on the outlet side is used to decrease the peak of the pressure trapped in the CVs; RG2 on the inlet side is used to restrain the cavitation.

As shown in **Figure 3**, the dashed line 1 means the flow between CV_8^{g2} and RG1 is forming. Solid line 2 and 4 mean that there is a fluid flow between CV_8^{g1} and CV_7^{g2} through RG1. Dashed line 3 means that the channel between CV_7^{g1} and RG1 is being turned off.

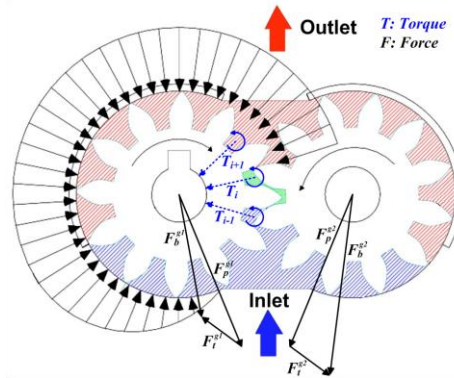


Figure 4 Calculated Forces & Torques Acting on Gear 1

Figure 4 shows the calculated Forces and Torques acting on the driving gear, where F_p^{g1} is the resultant force of the pressure distributed along the circle, F_c^{g1} is the contact force, F_b^{g1} is the force load on the bearing of gear1 and T_i is the torque supplied by the pressure in CVi.

Note: most of CVs, except those in the meshing zone, do not have any effect on the gear rotation, because the pressures in these tooth gaps counteract and no torque is acting (for example, $T_{i+1} = 0$). Details will be given in the next few sections.

The coordination system is shown in **Figure 5**.

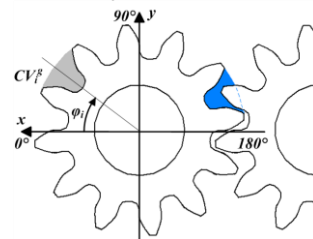


Figure 5 Coordination System

3.2 Control Volumes

3.2.1 Volume

A Control Volume is the space between two adjacent teeth. The maximum volume (see **Figure 5**, the gray region) of this gap is related to the gear dimensions (e.g. tooth number, modulus, thickness, etc). As the gears rotating, gear pairs mesh together. A tooth on the other gear squeezes into the gap, so the effective volume of the gap decreases (see **Figure 5**, the blue region). The effective volume of CVi is a function of the angle φ :

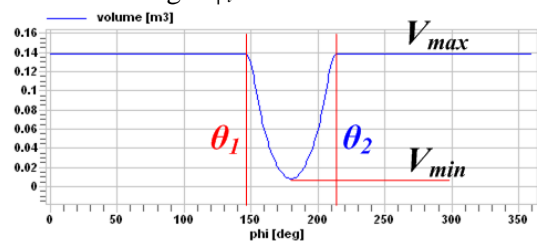


Figure 6 Effective Volume as a Function of Angle φ

This relationship can be expressed in an approximate equation:

$$V = \begin{cases} V_{\max}, & \varphi < \pi - \theta_1 \text{ or } \varphi > \pi + \theta_2 \\ x_3 [x_2 x_1 (V_{\max} - V_{\min}) / 2 \times \\ \cos((\varphi - \pi)f) + (V_{\max} + V_{\min}) / 2] \end{cases}, \text{otherwise}$$

where

$$x_3 = c_{31} \cos((\varphi - \pi)f) + c_{32}$$

$$x_2 = c_{21} \cos((\varphi - \pi)f) + c_{22}$$

$$x_1 = c_{11} \cos((\varphi - \pi) \times 2f) + c_{12}$$

$$f = 2\pi / (\theta_1 + \theta_2)$$

$$c_{11} = 0.25 \quad c_{12} = 0.75 \quad c_{21} = 0.25$$

$$c_{22} = -0.75 \quad c_{31} = -0.4 \quad c_{32} = 0.6$$

$$\text{phase} = \begin{cases} 2\pi / n \times (\text{Index} - 1) & , \text{if gear 1} \\ 2\pi / n \times (\text{Index} - 0.5) & , \text{if not} \end{cases}$$

θ_1 and θ_2 are the critical angles (see **Figure 7**).

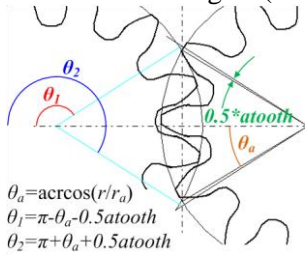


Figure 7 Critical Angles

3.2.2 Pressure distribution

The pressure in each CV_i is given by the equation:

$$\dot{p} = \frac{\beta}{V} \times (q + q_{\text{cham}})$$

where

β oil bulk modulus

q_{cham} the rate of the net flow that flows into the chamber

p pressure of the volume

The pressure distribution is shown in **Figure 4**.

3.2.3 Radial Force Produced by Pressure

As shown in **Figure 4**, the load on a bear is the sum of the loads produced by the distributing pressure and the contact force. Ideally, the loads due to the pressure of most CVs have only one radial component (F_r in **Figure 4**), since we have assumed (in section [错误!未找到引用源。](#)) that the pressure in any isolated region is well-proportioned.

When a CV turns into the meshing zone, the chamber will be divided into 2 or 3 parts by the coupling tooth on the other gear (see **Figure 8**). The

load due to the pressure in red zone 1 is equal to the sum of F_t and F_r . And

$$F_{t1} = B \times P_i^{g2} \times (r_m - r_a)$$

$$F_{r1} = B \times P_i^{g2} \times (r_m * \gamma_1)$$

Similarly, for zone 2 and zone 3 we have:

$$F_{t2} = B \times P_i^{g1} \times (r_n - r_m)$$

$$F_{r2} = B \times P_i^{g1} \times (r_m * \gamma_2)$$

$$F_{t3} = B \times P_{i-1}^{g2} \times (r_a - r_n)$$

$$F_{r3} = B \times P_{i-1}^{g2} \times (r_n * \gamma_1)$$

Then:

$$F_i = \sum_{j=1}^3 F_{tj} + \sum_{i=1}^3 F_{rj}$$

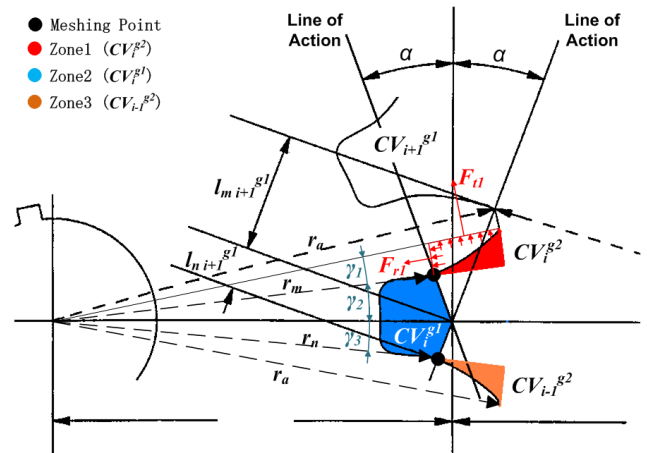


Figure 8 The chamber of CV_i^{g1} divided into 3 parts

3.2.4 Torque

Because only the component F_t has an effect on the gear rotation, the torque produced by CV_i^{g1} can be calculated as follows:

$$T = (F_{t1}(r_a + r_m) + F_{t2}(r_m + r_n) + F_{t3}(r_n + r_a)) / 2$$

Interfaces and Icon

The interfaces and icon are shown in **Figure 9**.

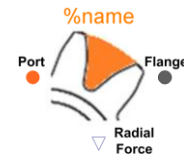


Figure 9 Icon of a Control Volume

3.3 Internal Leakages

Both gears rotate within the housing. The inter-space is made because of lubrication between two relative moving objects. The gap between two gear teeth and casing is assumed to maintain a known value. Leakage occurs through these internal clearances due to both the pressure differences and the relative motion.

There are four types of sources of internal leakages due to the pressure differences:

Q_p : Flows from one CV to an adjacent one through the tips of the gear teeth and the case.

Q_z : Flows from the outlet volume to the inlet volume through the meshing zone.

Q_m : Flows through the gap between two mating surfaces.

Q_d : Flows from the CVs to the drainage circle through the interspace between the gear and the case.

There is another kind of leakage that is caused by the relative motion and oil viscosity (see **Figure 10**, Q_w). It results in a little increase in the flow rate because the rotation brings oil to the outlet volume.

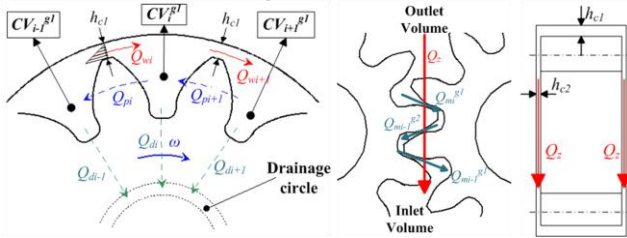


Figure 10 Internal Leakages

In general, for the first type of leakage it is assumed that the pressure around the case rises linearly across each tooth tip. Hence, the volumetric flow from CV_i^{gl} to CV_{i-1}^{gl} due to the pressure drop is given by ([2]):

$$Q_{pi} = dp \times G = dp \times h^3 b / (6\rho\nu\eta r)$$

where dp is the pressure differential.

The diagram of leakage Q_p is shown in **Figure 11**.

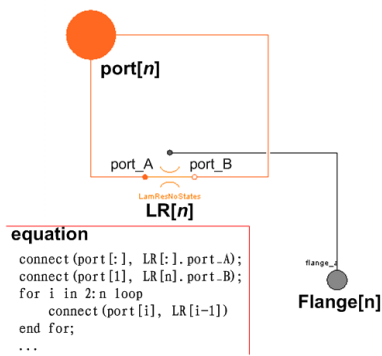


Figure 11 Diagram of Leakage Q_p

where n is the tooth number, LR is a laminar resistor model [9].

Other leakages due to pressure are similar to Q_{pi} , see [2][4].

The volumetric flow produced by the relative motion has a uniform distribution from zero to v , where v is the relative velocity of the tip with respect to the case. The volumetric flow is defined as [2]:

$$Q_{wi} = h_{cl} \times B \times v / 2$$

The interfaces and icons of five types of leakages are shown in **Figure 12**.

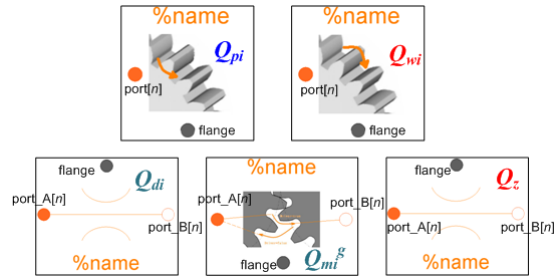


Figure 12 Icons of the Five Types of Leakages

3.4 Mechanical Parts

Mechanical parts contain the mounting, the bears and the mesh models. Mounting is used to fix the bears that support the gear pair. This paper does not cover the position of the shaft. So we choose a component FixedTranslation from Modelica.Mutibody to fix the two gears. Model IdealGear is used to transmit torque, which does not consider elasticity, damping or backlash. If we want to consider these effects, the gears have to be connected to other elements in an appropriate way.

The diagram of mechanical parts is shown in **Figure 13**.

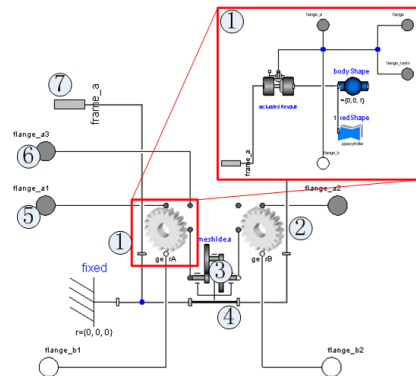


Figure 13 Diagram of the Mechanical Parts

- 1-Driving gear
- 2-Driven gear
- 3-IdealMesh
- 4-FixedTranslation
- 5-flange that transmits a driving torque
- 6-flange that transmits the resistance moment
- 7-frame, forces on shaft

Interfaces and icon of the mechanical parts are shown as follows:

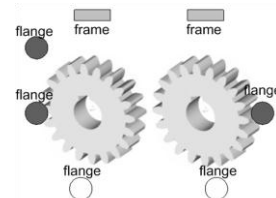


Figure 14 Interfaces and Icon of the Mechanical Parts

3.5 Radial Forces

Radial Forces produced by the CVs has been calculated (see section 3.2.3). In the model of CV, radial force is decomposed into x- and y- components, which are then summed up in the model RadialForce. **Figure 15** shows the diagram of the RadialForce as well as its interfaces and icon.

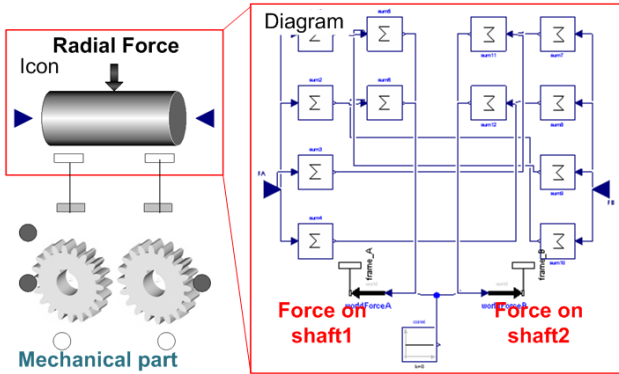


Figure 15 Interfaces, Icon and Diagram of Model RadialForce

3.6 Channels from CVs to Inlet/Outlet Volumes

Along with the rotation of the gears, when a CV comes to the position of CV_{i+2}^{gl} , the channel between it and the outlet volume opens; when it comes to the position of CV_i^{gl} , this channel becomes to close. Critical angles, shown in **Figure 16**, are calculated in respective models.

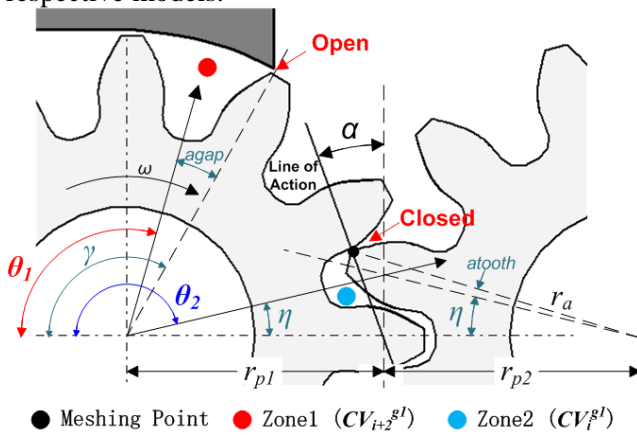


Figure 16 Critical Angles of Channel Model Connecting CV and Outlet Volume

Equations of the critical angles are given as follows:

$$\theta_1 = \gamma + a_{gap}$$

$$\theta_2 = \pi - \eta$$

where γ is a parameter about the pump dimension. η can be calculated from **Figure 16**.

The hydraulic diameter of the channel connecting a CV to the outlet volume is given by:

$$hd = \begin{cases} d_{max} & , \text{if } \theta_1 < \varphi < \theta_2 \\ d_{min} & , \text{otherwise} \end{cases}$$

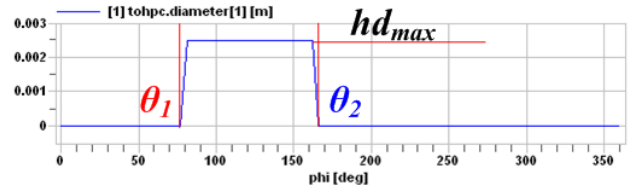


Figure 17 Hydraulic-Diameter as a Function of Angle φ

Interfaces and icons of a channel are shown in **Figure 18**.

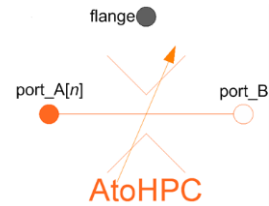


Figure 18 Interfaces and Icon of a Model connected to Outlet Volume (High Pressure Chamber)

The channel between a CV and the inlet volume is similar.

3.7 Viscosity Torque

In the ideal case, Viscosity models can be treated as dampers. There are two types of viscosity torques acting on a gear, the radial torque and the axial torque (see **Figure 19**). The equations of these torques are given by:

$$\tau_r = \rho v A \omega r^2 / h_{c1}$$

$$\tau_a = \frac{2\rho v}{h_{c2}} \int_{r_0}^{r_a} \omega r^2 2\pi r dr$$

$$= 4\pi\omega r^3 \rho v / (3h_{c2}) \quad , \text{if } r \leq r_0$$

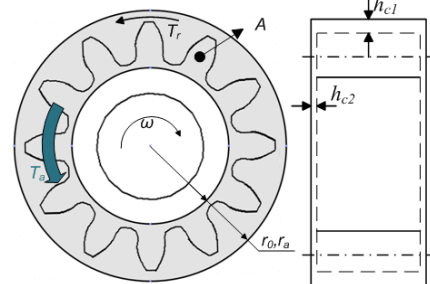


Figure 19 Radial Torques and Axial Torques

Interfaces and icons of the torque models are shown in **Figure 20**.

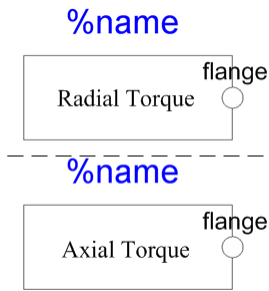
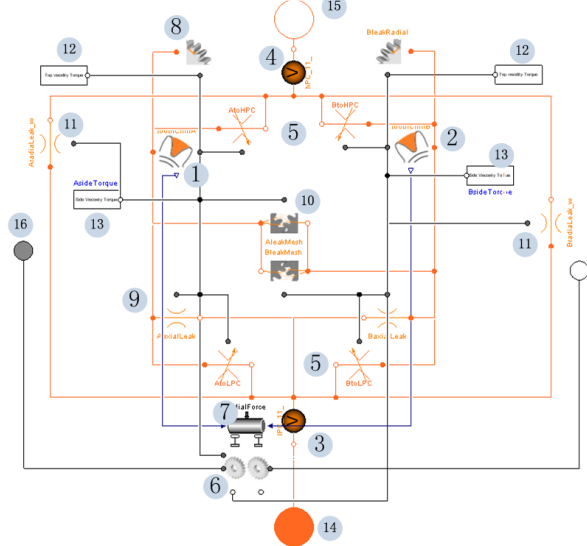


Figure 20 Interfaces and Icons of Torque Models

Other resistance moments, for example those caused by the bears, can be added as a component.

3.8 System Model

Combining all the components above, we obtain a whole pump as follows:



- 1-CVs on gear1 2-CVs on gear2 3-inlet vol 4-outlet vol
- 5-Channels(to in/outletChm) 6-Mechanical part 7-adialForce(optional)
- 8~11-Leakages 12~13-ViscosityTorques
- 14-inlet port 15-outlet port 16-shaft of gear1

Figure 21 Diagram of a Whole Pump

The interfaces and icon are shown in Figure 22.

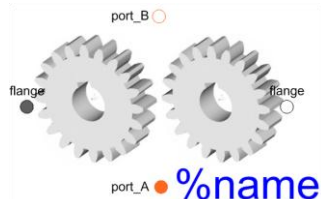


Figure 22 Interfaces and Icon of a Pump Model

4 Simulation Results

We implement the simulation based on the model shown in Figure 23 to enable the comparison between our results and the experimental results in [7].

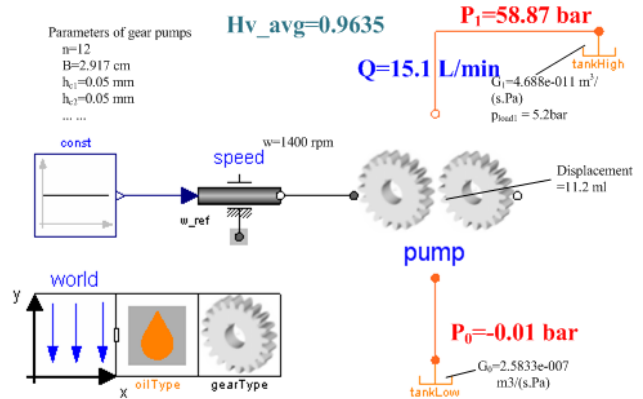


Figure 23 Simulation system diagram

The simulation curves of the volumetric flow ripple as well as the average flow rate (in L/min) is shown in Figure 24.

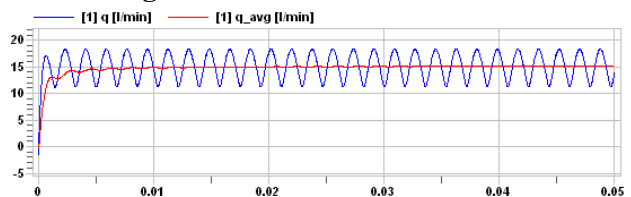


Figure 24 Volumetric flow ripple and average flow rate

The pressure distribution in a tooth space within a complete rotation cycle is shown in Figure 25.

Note: the pressure distribution in Figure 25 is very similar to the pressure distribution along the circle of a gear. But actually they are not the same.

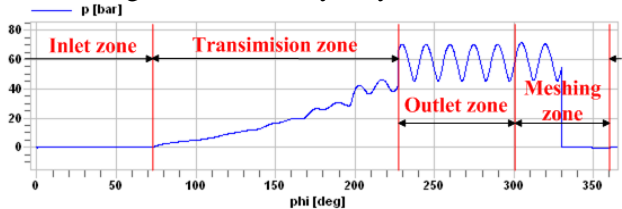


Figure 25 Pressure Distribution in a Tooth Space for a Complete Rotation Cycle

Order:

$$p_x = -(P + 1e6) \times \cos(\varphi)$$

$$p_y = (P + 1e6) \times \sin(\varphi)$$

The pressure distribution above is shown in Figure 26.

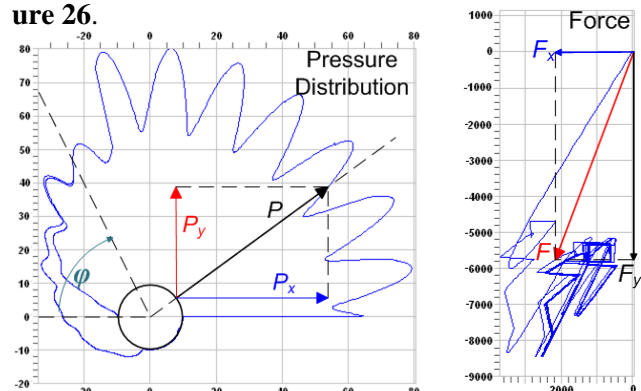


Figure 26 (Left) Pressure distribution in a Tooth Space for a Complete Rotation Cycle

The radius of the unit circle at the bottom is 10 bars.

Figure 27 (Right) Radial Force on Shaft 1 for a Complete Rotation Cycle and its Components in the Coordination in Figure 5.

By changing the parameters (e.g. G_0 , G_I shown in **Figure 23**), we obtain a set of predictions. Both the simulation results and the experimental results from [7] are listed in **Figure 28****Figure 29**. It is clear that the simulation and the experimental results are very close.

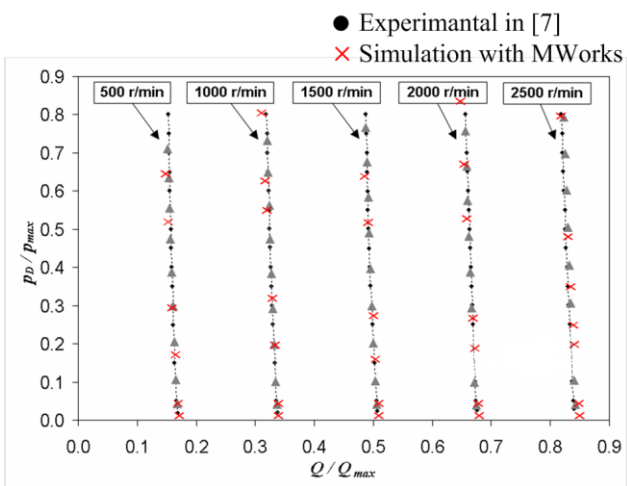


Figure 28 Dimensionless Characteristics of the Pump in Steady States

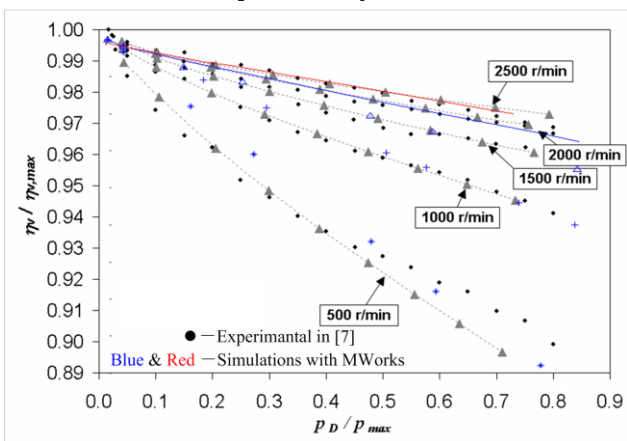


Figure 29 Dimensionless Values of the Volumetric Efficiency of the pump in Steady Conditions

5 Conclusions and Future Work

We present a new method of modeling the hybrid external gear pumps based on Modelica and assess it via simulation. The model takes into account the flow ripple, pressure distribution, leakages, meshing conditions, etc. Details of the components that make up the whole pump are introduced. Comparisons be-

tween the result and the experimental data are given in the end. Good agreement between them is founded.

From the pressure distribution in the space of a gear tooth, radial force on shaft can be easily calculated, on which the motion of the shaft can be simulated based. This is left for future work.

Another weakness of the study is that there is little comparison with real applications and measurement data. It is a drawback as some minor malfunctions and parameters probably could not be identified. Such comparison is also left for future work.

Acknowledgments

The paper is supported by National Nature Science Foundation of China (No.60874064, No.60704019, No.60736019), Key Project of National High Technology Research and Development Program (2009AA044501).

References

- [1] Noah D.Manring, Suresh B.Kasaragadda. The Theoretical Flow Ripple of an External Gear Pump. Journal of Dynamic Systems, Measurement, and Control. Transaction of the ASME 2003, September: 396-404
- [2] Aaron S.Heisler, John J.Moskwa, Frank J.Fronzak. SIMULATED HELICAL GEAR PUMP ANALYSIS USING A NEW CFD APPROACH. Fluids Engineering Division Summer Meeting 2009-78472
- [3] Abdul Wahab. ANALYTICAL PREDICTION TECHNIQUE FOR INTERNAL LEAKAGE IN AN EXTERNAL GEAR PUMP. Proceedings of ASME Turbo Expo,GT 2009-59287
- [4] R.Castilla, M.Gutes, P.J.Gamez-Monter, E.Codina. Experimental Study of the Shaft Motion in the Journal Bearing of a Gear Pump. Journal of Engineering for Gas Turbines and Power, 2009-052502
- [5] R.Castilla, M.Gutes, P.J.Gamez-Monter, E.Codina. Numerical Analysis of the Shaft Motion in the Journal Bearing of a Gear Pump. Journal of Engineering for Gas Turbines and Power, 2010-012504
- [6] G.Dalpia, G.D'Elia, E.Mucchi, A.Fernandez der Rincon. MODELING RUN IN PROCESS IN EXTERNAL GEAR PUMPS. Proceedings of ESDA2006-95466
- [7] Andrea Vacca, Germano Franzoni and Paolo Casoli. ON THE ANALYSIS OF EXPERIMENTAL DATA FOR EXTERNAL GEAR MACHINES AND THEIR COMPARISON

WITH SIMULATION RESULTS. International Mechanical Engineering Congress and Exposition, IMECE2007-422664

- [8] Fan-Li Zhou, Li-Ping Chen, Yi-Zhong Wu, Jian-Wan Ding, Jian-Jun Zhao, Yun-Qing Zhang. MWorks: a Modern IDE for Modeling and Simulation of Multidomain Physical Systems Based on Modelica. Modelica 2006, September 4th – 5th: 725-732
- [9] P.Beater. Modeling and Digital Simulation of Hydraulic Systems in Design and Engineering Education using Modelica and HyLib. Modelica Workshop 2000 Proceedings:33-40

Modeling and Simulation Vehicle Air Brake System

Li He, Xiaolong Wang, Yunqing Zhang, Jinglai Wu, Liping Chen
CAD Center, Huazhong University of Science and Technology, China
zhangyq@hust.edu.cn

Abstract

Air brake system has been widely used in heavy trucks and intercity buses for its great superiority over other brake system. The practical performance of air brake system may be greatly different from if we analyze it with static theory. Thus, it is necessary to build an integrate air brake system model to simulate the process of brake accurately. However, the dynamic mathematic model of air brake system is very complicated, which makes the model hard to be solved. In this paper, the components of air brake system are decomposed to several basic standard pneumatic components, and then build the system based on these basic standard pneumatic components. The standard pneumatic components which are built in the software-MWorks based on Modelica language include cylinder, nozzle, air reservoir, volume, and air pipe. An air brake system which contains brake valve, relay valve, brake chambers and pipelines is made based on the standard pneumatic components. The simulation results show the dynamic characteristics of air brake system.

Keywords: air brake system; dynamic model; Modelica

1 Introduction

Brake system affects the safety of automotive directly. The theories of brake system have been well studied [1, 2, 3]. However, the theories of brake system are almost limited to analyze the system's static characteristics such as the distribution of brake force between the front and rear axle, brake efficiency and so on. There is little literature focus on the dynamic characteristics of brake system.

The vehicle brake system can be generally divided into hydraulic brake system and air brake system. In fact, most tractor-trailer vehicles with a gross vehicle weight rating (GVWR) over 19000 lb, most single trucks with a GVWR over 31000 lb, most transit and intercity buses, and about half of all school buses are equipped with air brake systems [4]. Generally, the air brake system will show longer delay time than the hydraulic brake system for the response of air pressure is relative slowly to hydraulic pressure. In fact, the delay time for the pressure response has enormous influence on the performance of the vehicle brake system. Also the quick pressure response is one of the most important indispensable working conditions for the anti-brake system (ABS) which has been widely used in automobiles. Thus, to shorten the pressure response time is necessary. Roudlf [1] has given an empirical equation to calculate the delay time for the pressure response in air brake system, which correlated with the brake chamber volume and the length of pipelines. However, in practice we often need a more accurate model to research the delay time more exactly. So the model of air brake system which can accurately predict the pressure transmits from reservoir along the brake pipelines to the brake valve, relay valve and brake chamber should be build.

The mathematic model for dynamic responses of air brake system is very complicated since many differential equations need to be solved and many parameters may be correlated with each other, especially the parameters for the key components, such as brake valve, relay valve and pipelines. A non-linear model of the pneumatic subsystem of the air brake system which relates the pressure in the brake chamber to the brake valve plunger displacement and the supply pressure to the brake valve has been developed in [5,6]. The mathematic model is very compli-

cated, which leads it hard to be solved. Wu [7] studied the robust of pneumatic brake system in commercial vehicle, where the dynamics model is built in software-AMESim. In this paper, we use the Modelica language [8, 9] to construct the mathematic model, and we do not need to focus on the solver, which simplifies the work largely. In this study, the pneumatic brake system model are built and simulated based on the Modelica language. In section 2, the constitutions of air brake system are presented; the air brake system model is introduced in section 3. The simulation results are shown in section 4, and some conclusions are presented in the last section.

2 Air brake system components

An essential layout of the air brake system of a commercial vehicle is shown in Fig.1. The air brake system contains air compressor, storage reservoirs, brake valve, relay valve, brake chambers and some brake lines.

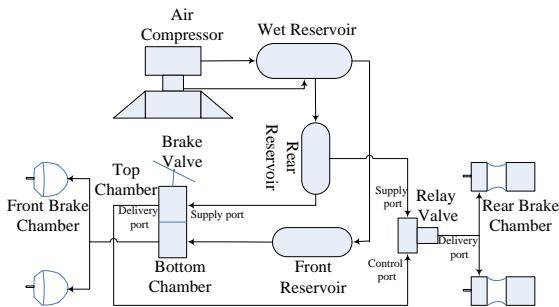


Figure. 1 Schematic representation of an air brake system

The driver applies the brake by pressing the brake pedal on the brake valve. This action makes the compressed air flow from the supply port of the brake valve to its delivery port. For the front circuit, the compressed air travels from the delivery port in bottom chamber of the brake valve through pipelines to the front brake chambers mounted on the front axles. For the rear circuit, the compressed air travels from the delivery port in top chamber of the brake valve through pipelines to the control port of relay valve, which opens the supply port of relay valve and makes the air flow to the rear brake chambers mounted on the front axles.

2.1 Brake Valve

The structure of brake valve is shown in Fig. 2.

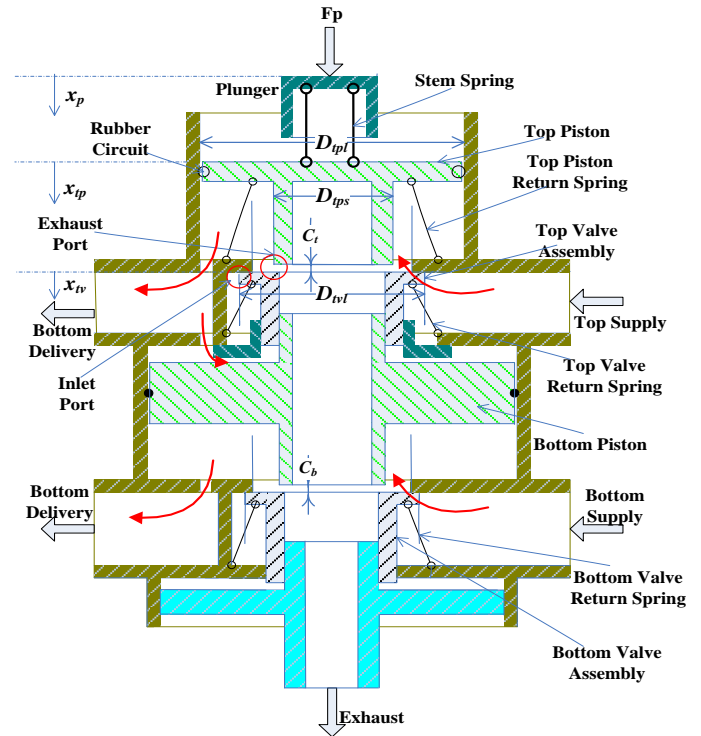


Figure. 2 Structure of brake valve

There are two supply ports, two delivery ports, and an exhaust port in the brake valve. The top supply port and bottom supply port are connected to rear reservoir and front reservoir respectively, while the top delivery port and bottom delivery port are connected to relay valve and front chambers respectively. For the top chamber, when the driver applies pedal force to the plunger, the force transmits to top piston compressing the rubber plunger spring. If the force acted on top piston is larger than the preload and friction, the top piston will move down. The exhaust port will be closed when the top piston touches the top valve, and the inlet port will open if the force transmitted to top valve can overcome the preload on top valve. Then the compressed air in rear reservoir can flows into top chamber, which makes the pressure in top chamber increasing. The compressed air in top chamber will produce a force that makes the top piston and top valve move up, and the inlet port will be closed gradually. The pressure keeps constant for both the inlet port and exhaust port are closed, thus the air pressure in top chamber

keeps constant. If the driver reduces pedal force, the balance of top piston will be broken. The top piston moves up for the force made by compressed air larger than pedal force, which make the exhaust port opened. Thus, the compressed air flow to atmosphere and reduces the pressure. The apply process of bottom chamber is similar to the top chamber.

The flow areas of inlet port and exhaust port have important effect on the characteristics of brake valve. The positions of top piston and top valve determine the flow areas of inlet port and exhaust port, expressed by the following equations.

$$A_m = \begin{cases} 0 & x_{ip} \leq C_t \\ \pi D_{ivi} (x_{ip} - C_t) & x_{ip} > C_t \end{cases} \quad (1)$$

$$A_{ex} = \begin{cases} \pi D_{ips} (C_t - x_{ip}) & x_{ip} \leq C_t \\ 0 & x_{ip} > C_t \end{cases} \quad (2)$$

Where C_t denotes the exhaust clearance, D_{ivi} and D_{ips} are the geometry dimensions shown in Fig. 2, x_{ip} is the displacement of top piston.

2.2 Relay Valve

The structure of relay valve is similar to the brake valve, shown in Fig. 3.

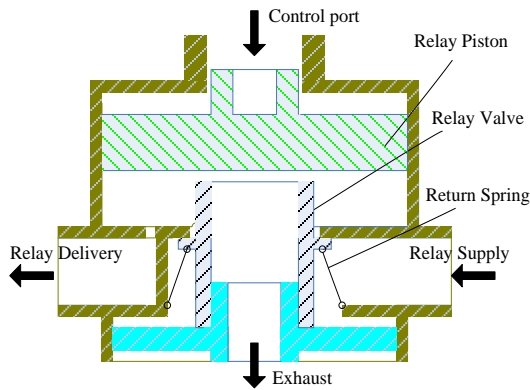


Figure. 3 Structure of relay valve

The difference between the brake valve and relay valve is that the control signal for brake valve is the force supplied by pedal while the control signal for relay valve is the compressed air from the brake valve. Relay valve can shorten the delay time for the rear brake chambers which are far from the brake valve.

2.3 Brake Chamber

Brake chamber is the component transiting air pressure to mechanical force. When the compressed air flows to the brake chamber, the air pressure pushes the push rod move, and then the mechanical force is transmitted to brake drum which makes the vehicle decelerated. In this paper, we simplify the rear service and spring brake chamber as the front service brake chamber (see Fig. 4).

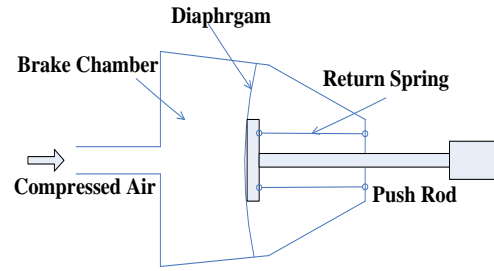


Figure. 4 Structure of brake chamber

3 Modeling vehicle air brake system

3.1 Standard pneumatic components

The standard pneumatic components include air volume, cylinder, nozzle, and pipe. We assume that the air temperature in volume is constant, and the thermal dynamic process is called isothermal. When the air flows to a constant volume, the pressure in the volume can be described by:

$$P = mRT/V \quad (3)$$

Where P denotes the pressure, m denotes the air mass in the volume, R denotes the ideal gas constant, T is the temperature, and V is the volume.

The cylinder is still assumed as isothermal, but the volume is not constant. The volume can be expressed as

$$V = V_0 + A_c s_p \quad (4)$$

Where V_0 denotes the initial volume of cylinder chamber, A_c is the effective area of piston, and s_p denotes the displacement of piston.

The nozzles in pneumatic system can be modeled as restrictions. The air flows from high-pressure chamber (upstream) to low-pressure chamber (downstream), the mass flow rate for the nozzle can be expressed by

$$\dot{m} = AC_q C_m \frac{P_u}{\sqrt{T_u}} \quad (5)$$

Here, A is the flow area of nozzle, and C_q is the flow coefficient which can be expressed as

$$C_q = 0.8414 - 0.1002 \left(\frac{P_d}{P_u}\right) + 0.8415 \left(\frac{P_d}{P_u}\right)^2 - 3.9 \left(\frac{P_d}{P_u}\right)^3 + 4.6001 \left(\frac{P_d}{P_u}\right)^4 - 1.6827 \left(\frac{P_d}{P_u}\right)^5 \quad (6)$$

C_m is the flow parameter can be expressed by [7, 8,10,11]

$$C_m = \begin{cases} \sqrt{\frac{2}{R} \cdot \frac{\gamma}{\gamma-1}} \cdot \sqrt{\left(\frac{P_d}{P_u}\right)^2 - \left(\frac{P_d}{P_u}\right)^{\frac{\gamma+1}{\gamma}}}, & \text{if } \frac{P_d}{P_u} > P_{cr} = \left(\frac{2}{\gamma+1}\right)^{\frac{\gamma}{\gamma-1}} \\ \sqrt{\frac{2}{R} \cdot \frac{\gamma}{\gamma-1}} \cdot \left(\frac{2}{\gamma+1}\right)^{\frac{1}{\gamma-1}}, & \text{if } \frac{P_d}{P_u} \leq P_{cr} = \left(\frac{2}{\gamma+1}\right)^{\frac{\gamma}{\gamma-1}} \end{cases} \quad (7)$$

Where γ denotes the specific heat ratio, P_{cr} denotes critical pressure ratio, P and T denote the pressure and temperature respectively, subscript u and d denote upstream and downstream respectively. The pipelines can also be modeled as restriction.

3.2 Air brake components

Build the air brake components models based on the standard pneumatic model in software-MWorks. The top chamber of brake valve is shown in Fig. 5.

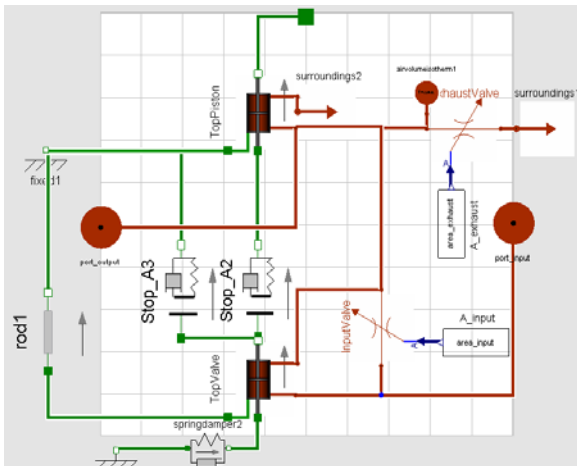


Figure. 5 Structure of top chamber

The bottom chamber of brake valve is shown in Fig. 6.

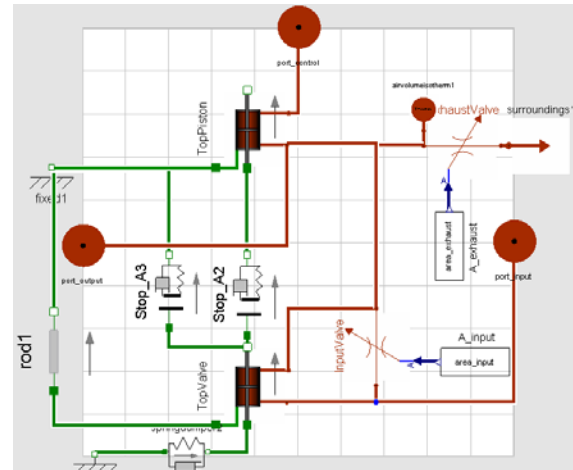


Figure. 6 Structure of bottom chamber

The structure of relay valve is the same as the bottom chamber of brake valve, so we would not repeat it. Build the brake chamber based on the standard pneumatic model, shown in Fig. 7.

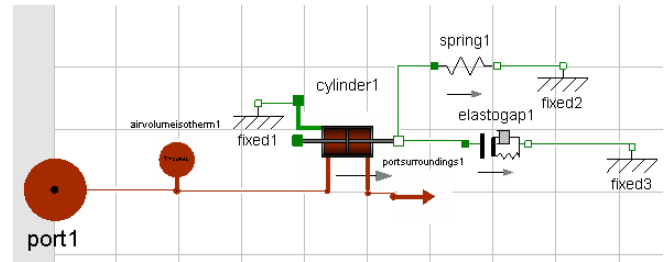


Figure. 7(a) Structure of brake chamber

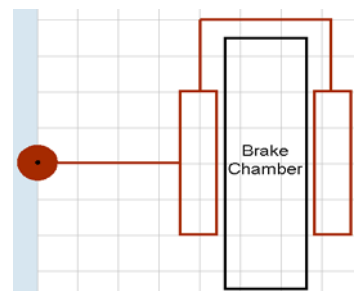


Figure. 7(b) Icon of brake chamber

3.3 Air brake system

Based on the components of air brake system, construct the air brake system shown in Fig. 8.

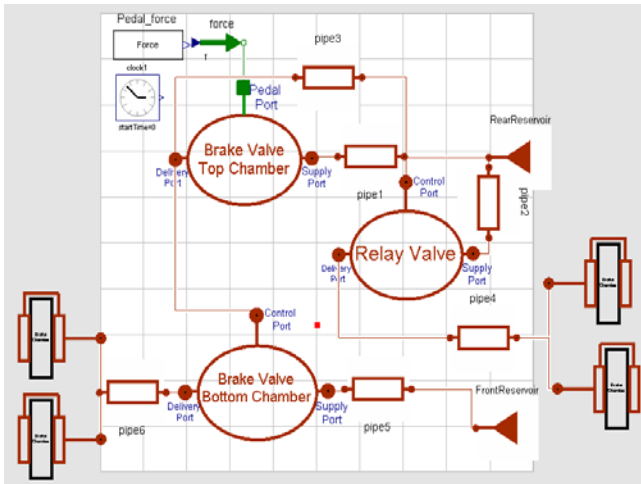


Figure. 8 Air brake system

4 Simulation results

Take the plunger force (enlarged pedal force) as input signal and the pressure of air in brake chamber as the output signal. Figure.9 shows a specified plunger force variation curve. The plunger force increases to 1200N from 0N during the period between 0 second and 0.4 second (apply phase), keep the force unchanged until the time at 4 second (hold phase), and reduce the force to 0N during the period between 4 second and 4.4 second (exhaust phase).

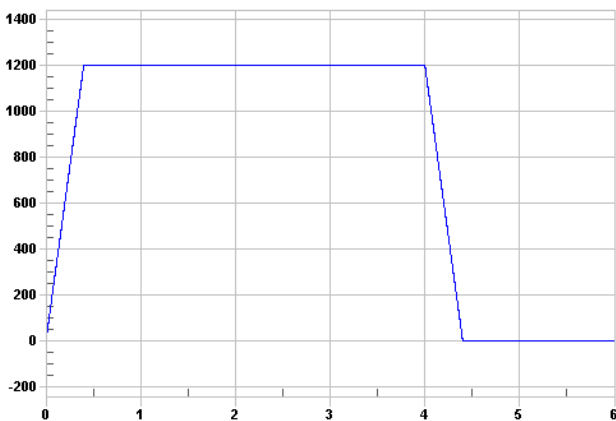


Figure. 9 Plunger force

The pressure in brake chambers are shown in Fig. 10. The blue line denotes the pressure in front brake chamber, and red line denotes the pressure in rear brake chamber.

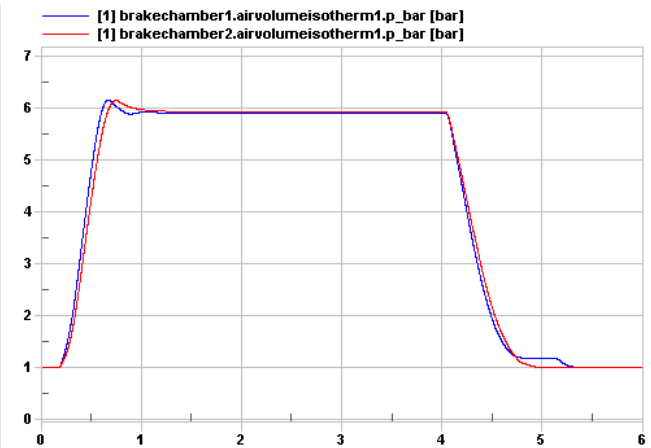


Figure. 10 Pressure in brake chamber

From the simulation results, we find that the pressure in brake chamber is consistent with the plunger force, but the pressure variation delays to the plunger force. Through the simulation, we can research the dynamic response of air brake system further.

5 Conclusions

The dynamic model of air brake system is presented in this paper. The mathematic models of some standard pneumatic components are introduced and built in software- MWorks. The key components of air brake system such as brake valve, relay valve, and brake chamber are constructed based on the standard pneumatic components. The integrate air brake system is also built using the key components, and the simulation results show dynamic characteristics of air brake system. Through the dynamic simulation, the delay time of brake system can be obtained more accurately, which has helpful to improve the performance of air brake system.

Acknowledgement

This work was supported by the National High-Tech R&D Program, China (No. 2009AA044501).

References

[1] Limpert. R. Brake Design and Safety. SAE Order No. R-198, 1999.

- [2] Limpert. R. Engineer Design Handbook, Analysis and Design of Automotive Brake Systems. US Army Material Development and Readiness Command, DARCOM-P - 706-358, 1976.
- [3] Bert. B, Karlheinz. H. Brake Technology Handbook. SAE Order No.R-367, 2008.
- [4] Williams. S. F, Knipling. R. R. Automatic Slack Adjusters for Heavy Vehicle Air Brake Systems. Nat. Highway Traffic Safety Administration, Washington, DC, Tech. Rep. DOT HS 807 724, Feb 1991.
- [5] Subramanian. S. C, Darbha. S, and Rajagopal. K. R. A Diagnostic System for Air Brakes in Commercial Vehicles. IEEE Transactions on Intelligent Transportation Systems, Vol.7, No.3, September 2006, pp360-376.
- [6] Subramanian. S. C, Darbha. S, and Rajagopal. K. R. Modeling the Pneumatic Subsystem of an S-cam Air Brake System. Trans. of the ASME, J. of Dynamic Systems, Measurement, and Control, Vol.126, 2004, pp36-46.
- [7] Wu. J, Zhang. H, Zhang. Y, Chen. L. Robust design of pneumatic brake system in commercial vehicles. SAE, 2009-01-0408.
- [8] Fritzson P., Vadim V. Modelica -- A Unified Object-Oriented Language for System Modeling and Simulation. Proceedings of the 12th European Conference on Object-Oriented Programming, 1998, pp.67 – 90.
- [9] Zhou F, Chen L. and Wu Yi., etc. MWorks: a Modern IDE for Modeling and Simulation of Multidomain Physical Systems Based on Modelica. Modelica 2006, September 4th – 5th, pp. 725-732.
- [10] Bowlin. C. L, Subramanian. S. C, Darbha. S, and Rajagopal. K. R. Pressure Control Scheme for Air Brakes in Commercial Vehicles. IEE Proc. Intelligent Transportation Systems, Vol. 153, No.1, March 2006, pp21-32.
- [11] Peter. B. Pneumatic Drives. Springer-Verlag Berlin Heidelberg, 2007.
- [12] Burrows C R, Peckham R G (1977) Dynamic characteristics of a pneumatic flapper valve. Journal Mechanical Engineering Science. 19(3):113–121.

A Modelica Library for Simulation of Electric Energy Storages

M. Einhorn¹ F. V. Conte¹ C. Kral¹ C. Niklas¹ H. Popp¹ J. Fleig²

¹AIT Austrian Institute of Technology
 Mobility Department, Electric Drive Technologies
 Giefinggasse 2, 1210 Vienna, Austria
markus.einhorn@ait.ac.at

²Vienna University of Technology
 Institute of Chemical Technologies and Analytics
 Getreidemarkt 9/164ec 1060 Vienna, Austria

Abstract

This article gives an overview of the Electric Energy Storage (EES) library, which is proposed for inclusion in the Modelica Standard Library. The library contains models with different complexity for simulating of electric energy storages like batteries (single cells as well as stacks) interacting with loads, battery management systems and charging devices. It is shown how the models are defined and how they can be parametrized. Finally, two example simulations are presented.

Keywords: Energy storages, library, battery simulation

1 Introduction

Simulation is a commonly used technique to reduce costs during the design and development process. The energy storage system is a key issue, especially for electric vehicles. Basic models of electric energy storages (EES) are already included in the commercial SmartElectricDrives (SED) library [1]. The EES library, presented in this article, provides basic as well as more complex models for battery cells and for battery stacks. It includes models for battery monitoring and measurement, chargers, loads, sensors and battery management. This library can be used to simulate the behavior of electric energy storages in mobile devices, stationary applications and in transportation systems including hybrid as well as electric vehicles. The models of the EES library are designed as universally as possible so that even very specific scenarios can be simulated by varying the parametrization. In the future it is intended to include the EES library in

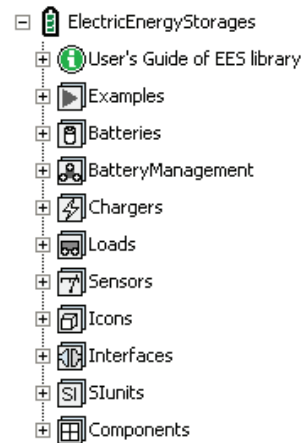


Figure 1: Electric Energy Storage (EES) library structure

the Modelica Standard Library (MSL).

This article shows how the EES library is structured, how the fundamental models are defined and which parameters are needed. Finally, two example simulations are presented. For implementing the EES library, Dymola 7.4 and Modelica 3.2 are used [2–4].

2 Library structure

The EES library is structured as shown in Fig. 1. The fundamental packages and models are now explained in more detail.

2.1 Batteries

The Batteries package contains models for cells as well as for stacks with n_s serially connected cells and n_p cells in parallel. Its structure is shown in Fig. 2.

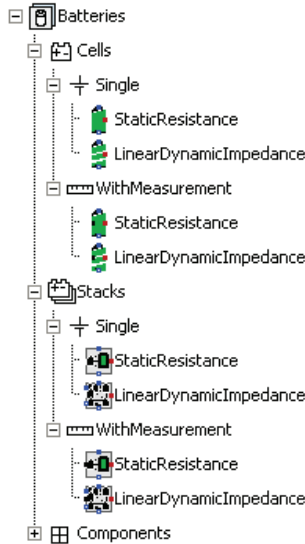


Figure 2: Structure of the Batteries package as a part of the EES library.

2.1.1 Cells

In the Cells package there are two different types of cell models: Single and WithMeasurement. While the Single cell models are models of the battery cell only, the WithMeasurement cell models extend the Single cell models with basic measurement. Each of these models can either be a simple cell model with just an ohmic impedance (StaticResistance) or a more complex cell model considering basic self discharge, a variable ohmic impedance and a variable number of variable RC elements (LinearDynamicImpedance).

Single

The StaticResistance single cell model as well as the LinearDynamicImpedance single model are shown in Fig. 3 and combined in the Single package.

Both single cell models have a positive (pin_p), a negative (pin_n) and an optional temperature connector (heatPort). If the heatPort is not used, it can be disabled and the model operates at a fixed temperature ($T_{operational}$). The common parameters of the StaticResistance and the LinearDynamicImpedance cell models are given in Table 1. For the StaticResistance single cell model additionally to Table 1, the parameter given in Table 2 is necessary. Instead, for the LinearDynamicImpedance single cell model, additionally to Table 1 the parameters given in Table 3 are necessary.

Table 1: Common input parameters of the StaticResistance and the LinearDynamicImpedance cell model.

name	unit	description
SOC_{ini}		initial state of charge
$OCVtable$	V	lookup table for the open circuit voltage OCV vs. the state of charge SOC
t_{total}	s	total cell life time
Q_{ini}	C	initial transferred charge
Q_{total}	C	total transferable charge
C_0	C	capacity at T_{ref} for $Q_{abs} = 0$ and $t = 0$
$k_{C t}$	C/s	linear t dependency of the capacity C
$k_{C Q_{abs}}$	C/C	linear Q_{abs} dependency of the capacity C
x_C		factor at which value of the capacity C $SOH_C = 0$
$useHeatPort$		boolean variable for using the heat port
$T_{operational}$	K	operational temperature if the heat port is not used
T_{ref}	K	reference temperature
$alphaR_s$	K^{-1}	linear temperature coefficient of R_s
$alphaC$	K^{-1}	linear temperature coefficient of the capacity C

Table 2: Additional input parameter of the StaticResistance cell model.

name	unit	description
R_{sref}	Ω	ohmic resistance at reference temperature T_{ref}

The output variables both for the StaticResistance single cell model and the LinearDynamicImpedance are given in Table 4 and the calculation of them are presented in the following.

Starting from SOC_{ini} the

$$SOC = SOC_{ini} - \frac{Q}{C}, \quad (1)$$

with the removed charge

$$Q = \int_{t_{start}}^{t_{stop}} I(t) dt. \quad (2)$$

The open circuit voltage OCV of a battery cell changes with SOC and can be extracted from a lookup table $OCVtable$ between the charging voltage limit CVL and the discharging voltage limit $DVLL$. This linearly interpolated lookup table for a lithium ion (Li-ion) battery cell [5] is exemplarily shown in Fig. 4.

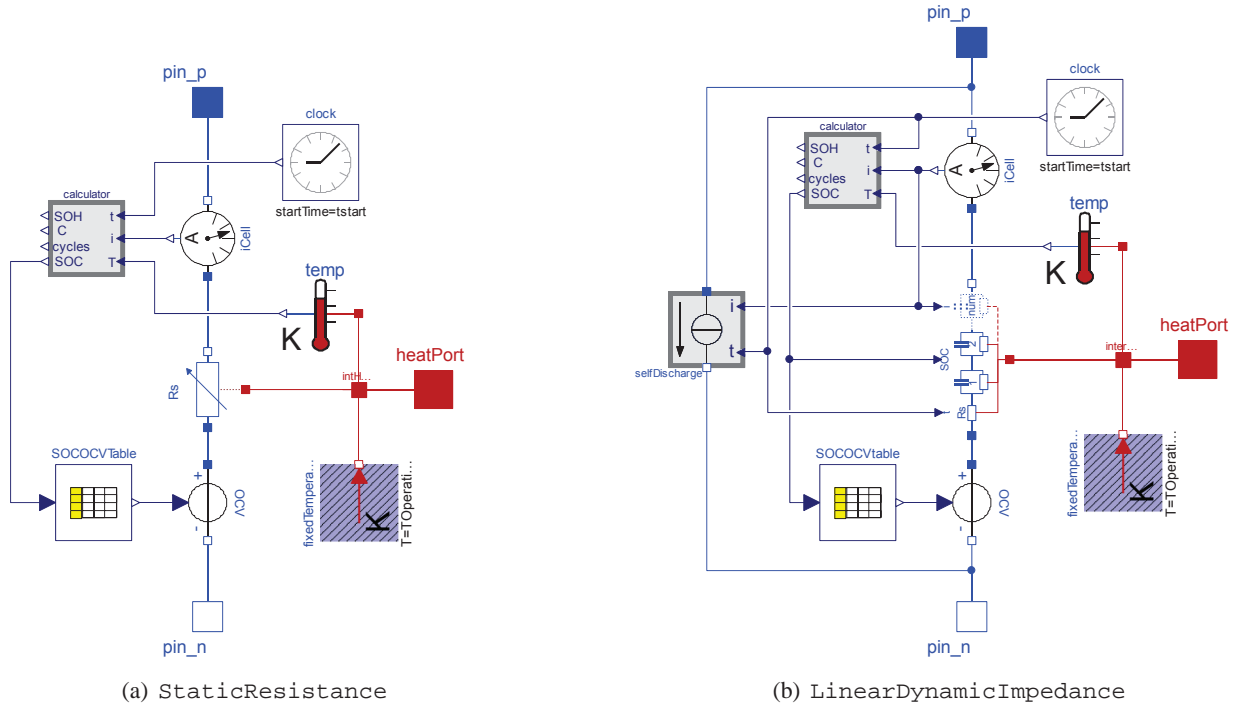


Figure 3: Single cell model with a static impedance (a) and with a variable number of variable RC elements, a variable ohmic impedance as well as with basic self discharge (b)

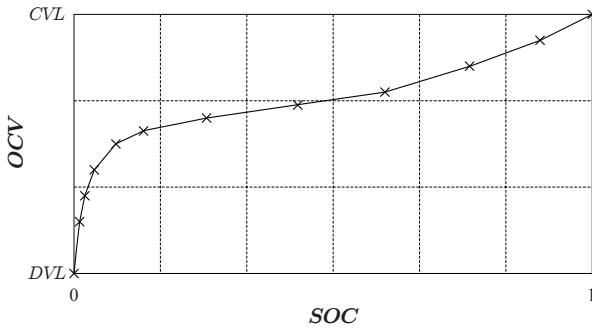


Figure 4: Linear interpolation of the measured open circuit voltage (OCV) for different state of charge (SOC) of a Li-ion battery cell [5].

The total transferred charge between t_{start} and t_{stop} is

$$Q_{abs} = Q_{ini} + \int_{t_{start}}^{t_{stop}} |I(t)| dt \quad (3)$$

and the equivalent number of cycles

$$cycles = cycles_{ini} + \int_{t_{start}}^{t_{stop}} \frac{|I(t)|}{2 \cdot C} dt, \quad (4)$$

with

$$cycles_{ini} = \frac{Q_{ini}}{2 \cdot \frac{C_0 + C(t_{start}, Q_{ini}, T_{heatPort})}{2}}, \quad (5)$$

relates the total transferred charge to the cell capacity C . Therefore, one cycle is equivalent to the charge transfer (regardless in which direction) of one full discharge and one full charge of the current capacity C .

Both cell models consider basic aging, which can be divided into calendaric aging and aging due to cycling. Calendaric aging of a cell is estimated from the time t and the absolute transferred charge Q_{abs} defines the aging due to cycling. Aging of a battery mainly influences the capacity C (decreasing) and the internal impedance (increasing).

The cell capacity

$$C = (C_0 + k_{Ct} \cdot t + k_{CQ_{abs}} \cdot Q_{abs}) \cdot (1 + \alpha C \cdot (T_{heatPort} - T_{ref})) \quad (6)$$

is temperature dependent and decreases with increasing time t (calendaric aging) as well as with increasing transferred charge Q_{abs} (aging due to cycling).

The difference between the `StaticResistance` and the `LinearDynamicImpedance` single cell model is the configuration of internal impedance on the one hand and the self discharge on the other hand.

The `StaticResistance` single cell model has just a single, temperature dependent, ohmic

Table 3: Additional input parameters of the `LinearDynamicImpedance` cell model.

name	unit	description
I_{sd0}	A	self discharge current at T_{ref} for $Q_{abs} = 0$ and $t = 0$
$k_{sd t}$	A/s	linear t dependency of self discharge current
$k_{sd Q_{abs}}$	A/C	linear Q_{abs} dependency of self discharge current
$alphasd$	K^{-1}	linear temperature coefficient of self discharge current I_{sd}
R_{s0}	Ω	series resistance at T_{ref} for $Q_{abs} = 0$ and $t = 0$
$k_{R_s SOC}$	Ω	linear SOC dependency of R_s
$k_{R_s t}$	Ω/C	linear t dependency of R_s
$k_{R_s Q_{abs}}$	Ω/C	linear Q_{abs} dependency of R_s
num		number of series RC elements
$\langle R_{d0} \rangle$	Ω	array of length num of R_d at T_{ref} for $SOC = 0$, $Q_{abs} = 0$ and $t = 0$
$\langle k_{R_d SOC} \rangle$	Ω	array of length num of linear SOC dependency of R_d
$\langle k_{R_d t} \rangle$	Ω/s	array of length num of linear t dependency of R_d
$\langle k_{R_d Q_{abs}} \rangle$	Ω/C	array of length num of linear Q_{abs} dependency of R_d
$\langle alphaR_d \rangle$	K^{-1}	array of length num of linear temperature coefficient of R_d
$\langle C_{d0} \rangle$	F	array of length num of C_d for $SOC = 0$, $Q_{abs} = 0$ and $t = 0$
$\langle k_{C_d SOC} \rangle$	F	array of length num of linear SOC dependency of C_d
$\langle k_{C_d t} \rangle$	F/s	array of length num of linear t dependency of C_d
$\langle k_{C_d Q_{abs}} \rangle$	F/C	array of length num of linear Q_{abs} dependency of C_d
x_Z		factor at which value of the internal, ohmic impedance Z $SOC_Z = 0$

impedance, modeled as

$$R_s = R_{sref} \cdot (1 + alphaR_s \cdot (T_{heatPort} - T_{ref})). \quad (7)$$

It does not consider impedance increase due to aging.

In contrast, the `LinearDynamicImpedance` single cell model has an ohmic impedance and num serially connected RC elements for the transient behavior of the electrodes of an electrochemical energy storage as shown in Fig. 5 [6, 7]. All ohmic impedances ($R_s, R_{d1} \dots R_{dnum}$) are temperature dependent and have a linear dependency on state of charge SOC , on the time t (calendaric aging) as well as on the transferred charge Q_{abs} (aging due to cy-

 Table 4: Calculated output variables of the `StaticResistance` and the `LinearDynamicImpedance` cell model.

name	unit	description
SOC		state of charge
OCV	V	open circuit voltage
Q_{abs}	C	total transferred charge
$cycles$		number of equivalent cycles
t	s	calendaric cell time
SOH		state of health
SOS		state of sickness
C	C	capacity
V	V	cell voltage

cling). The serial resistor R_s is modeled as

$$R_s = (R_{s0} + k_{R_s SOC} \cdot SOC + k_{R_s t} \cdot t + k_{R_s Q_{abs}} \cdot Q_{abs}) \cdot (1 + alphaR_s \cdot (T_{heatPort} - T_{ref})). \quad (8)$$

For $n = 1 \dots num$

$$R_d[n] = (R_{d0}[n] + k_{R_d SOC}[n] \cdot SOC + k_{R_d t}[n] \cdot t + k_{R_d Q_{abs}} \cdot Q_{abs}) \cdot (1 + alphaR_d[n] \cdot (T_{heatPort} - T_{ref})). \quad (9)$$

and since there is no temperature dependency for capacitances considered

$$C_d[n] = C_{d0}[n] + k_{C_d SOC}[n] \cdot SOC + k_{C_d t}[n] \cdot t + k_{C_d Q_{abs}}[n] \cdot Q_{abs}. \quad (10)$$

Moreover, the `LinearDynamicImpedance` single cell model considers basic self discharge which is linear dependent on the temperature, the time (calendaric aging) and the transferred charge (aging due to cycling).

While with the `StaticResistance` single cell model only the basic impedance behavior can be simulated, the `LinearDynamicImpedance` can be used to simulate single cells very accurate if the parametrization work is well done.

The capacity as well as the internal impedance (only with the `LinearDynamicImpedance` single cell model) can change due to aging and the state of health SOH (and accordingly the state of sickness SOS) compares the current condition of a battery cell to its ideal (initial) condition. The SOH is divided into the SOH_C and SOH_Z :

$$SOH = 1 - SOS = SOH_C \cdot SOH_Z, \quad (11)$$

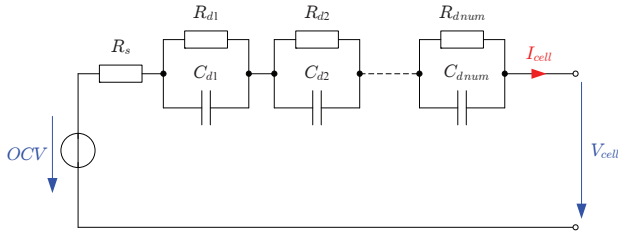


Figure 5: Battery model with one ohmic impedance and num serially connected RC elements.

as shown in Fig. 6.

For a new cell, $SOH_C = 1$ and $SOH_Z = 1$ and therefore also $SOH = 1$. When the capacity C decreases (due to calendaric aging or aging due to cycling) to $x_C \cdot C_0$ (e.g. with $x_C = 0.8$), $SOH_C = 0$. Hence,

$$SOH_C = \frac{1}{C_0 \cdot (1 - x_C)} \cdot C - \frac{x_C}{1 - x_C}. \quad (12)$$

Similarly, when the sum of all internal, ohmic impedances

$$Z = R_s + R_d[1] + R_d[2] + \dots + R_d[num] \quad (13)$$

increases (due to calendaric aging or aging due to cycling) to $x_Z \cdot Z_0$ (e.g. with $x_Z = 2$), where

$$Z_0 = R_{s0} + R_{d0}[1] + R_{d0}[2] + \dots + R_{d0}[num], \quad (14)$$

$SOH_Z = 0$. Therefore,

$$SOH_Z = \frac{1}{Z_0 \cdot (1 - x_Z)} \cdot Z - \frac{x_Z}{1 - x_Z}. \quad (15)$$

Fig.7 shows the dependency of SOH_C and SOH_Z from C and Z , respectively.

WithMeasurement

In the `WithMeasurement` package there are two cell models with measurement: The `StaticResistance` and the `LinearDynamicImpedance`. Both contain basic measurement and instances of the corresponding single cell model. Fig. 8 shows for example the `LinearDynamicImpedance` with measurement. The voltage, the current and the temperature of the single cell model are measured and provided with the `singleCellBus` (cf. section 2.7). The cell models with measurement have the same connectors as the single cell models (`pin_p`, `pin_n` and `heatPort`) but have additionally the `singleCellBus`. Therefore several instances of these can be connected

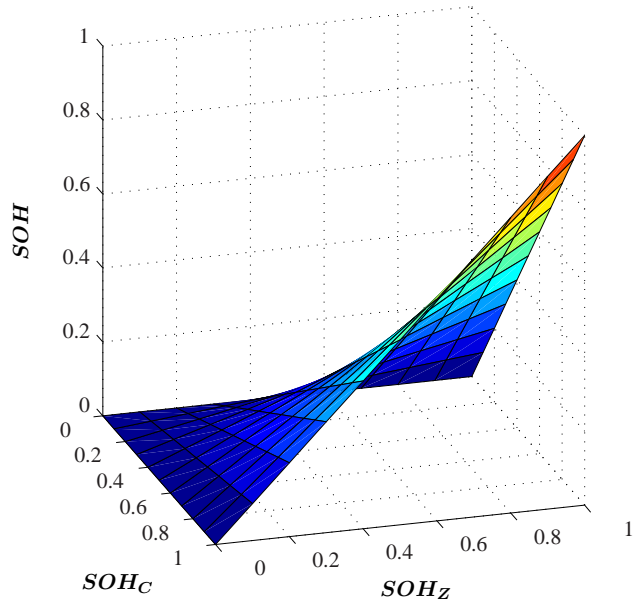


Figure 6: Dependency of the state of health SOH from SOH_C and SOH_Z .

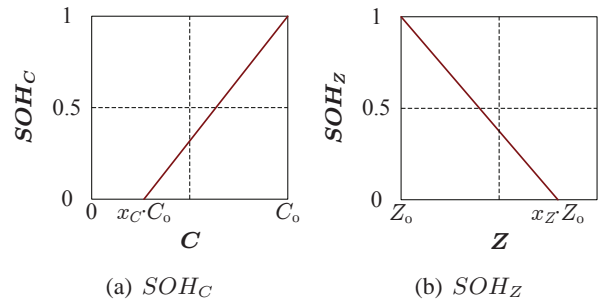


Figure 7: Partial state of health with respect to the capacity C and to the impedance Z with parameters $0 < x_C < 1$ and $x_Z > 1$.

together (serially and in parallel) and the basic measurement of each cell is done.

The advantage of separating the cell models in `Single` and `WithMeasurement` models shows up when it comes to stacks and the current of each cell should be measured. Fig. 9 shows the icons for the `Single` and `WithMeasurement` for the `StaticResistance` cell as well as for the `LinearDynamicImpedance` cell models with enabled `heatPort`.

2.1.2 Stacks

Stacks are n_s serially connected cells and n_p cells in parallel as shown in Fig. 10. The `Stacks` package is structured in the same way as the `Cells` package. There are `Single` and `WithMeasurement` stacks where each uses either the `StaticResistance` or

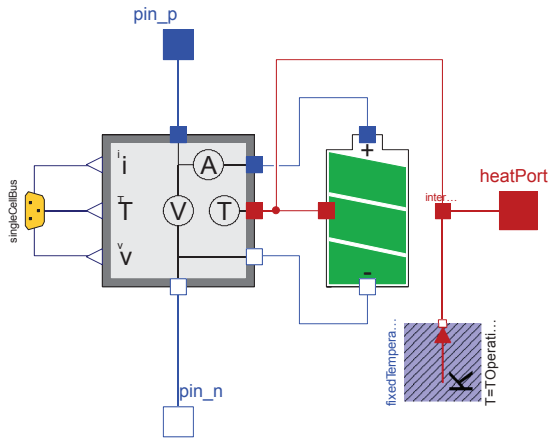


Figure 8: Cell model with measurement. The measured current, temperature and voltage are provided to the outside via the singleCellBus

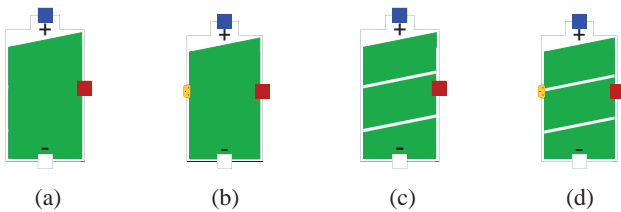


Figure 9: Icons for the StaticResistance (Single (a), WithMeasurement (b)) and for the LinearDynamicImpedance (Single (c), WithMeasurement (d)) cell model with enabled heat port.

the LinearDynamicImpedance cell models.

The StaticResistance single stack model is a stack with all equal cells and therefore only one cell needs to be calculated and parameterized. It is basically a StaticResistance single cell model that scales the parameters of the model components and behaves like $n_s \cdot n_p$ equally parameterized instances of StaticResistance single cell models. For example the value of the series resistance is then $R_{s0} \cdot n_s / n_p$ with the value R_{s0} for a single cell. Hence, it is much faster than a model with $n_s \cdot n_p$ instances of equally parametrized cell models.

The LinearDynamicImpedance single stack model has $n_s \cdot n_p$ instances of the LinearDynamicImpedance single cell model and the serial and parallel connections are textually generated with loops:

```
equation
//series connection
for s in 1:ns-1 loop
  connect(cell[s,1].pin_n,cell[s+1,1].pin_p);
end for;
```

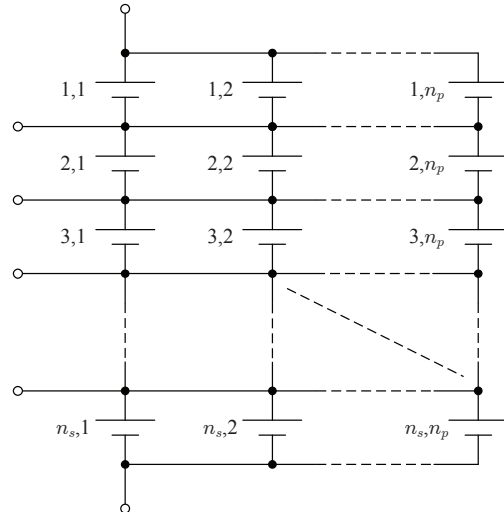


Figure 10: Battery stack with n_s serially and n_p connected cells.

```
//parallel connection
for p in 1:np-1 loop
  for s in 1:ns loop
    connect(cell[s,p].pin_p,cell[s,p+1].pin_p);
    connect(cell[s,p].pin_n,cell[s,p+1].pin_n);
  end for;
end for;
//connector connection
for s in 1:ns loop
  connect(cell[s,1].pin_p,pin_pCell[s]);
  connect(cell[s,1].pin_n,pin_nCell[s]);
end for;
//top connection
connect(cell[1,1].pin_p,pin_pPackage);
//bottom connection
connect(cell[ns,np].pin_n,pin_nPackage);
//heatPort connection
connect(cell[:,:].heatPort,heatPort[:,:]);
```

Each scalar parameter of the LinearDynamicImpedance single cell model is extended to an array of dimension $n_s \times n_p$ and the array parameters of the LinearDynamicImpedance single cell model now have the dimension $n_s \times n_p \times num$. Each cell in this stack can be parametrized individually and therefore also cell variance in a stack can be simulated. All single cell connectors and all temperature connector are conditionally available. Fig. 11 shows the icons for all different stack models. Therefore a separate thermal model can be considered. With the single cell connectors and an additional model even cell balancing can be simulated [8].

From each single stack model (StaticResistance and LinearDynamicImpedance) there is also a version with measurement (StaticResistance

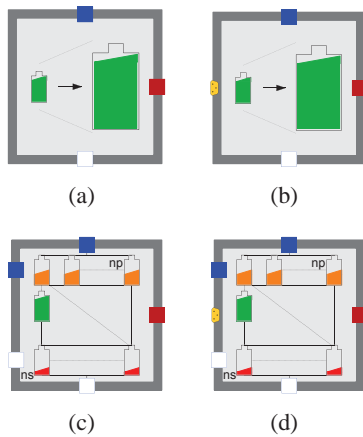


Figure 11: Icons for the Single (a), WithMeasurement (b) StaticResistance stack models with enabled heat ports and the Single (c), WithMeasurement (d) LinearDynamicImpedance stack models with enabled heat as well as single cell ports .

Table 5: Parameters of the VoltageCycling battery management model.

name	unit	description
n_s		number of serially connected cells
n_p		number of parallel connected cells
I_{final}	A	final charging switch off current
V_{max}	V	maximal cell voltage
V_{min}	V	minimal cell voltage
t_c	s	delay time after charging
t_d	s	delay time after discharging
ini		true for initial discharging

and LinearDynamicImpedance) in the Stacks package.

2.2 Battery Management

In the current version of the EES library there is a voltage cycling device (VoltageCycling) implemented. Fig. 12 gives an overview about the VoltageCycling model and its parameters are given in Table 5

It basically has two boolean outputs to operate with loads and charging devices (cf. section 2.3 and 2.4). The cellBus can either be connected to a cell or a stack. During charging if V_{max} of any cell is reached and the charging current of all parallel connected cells is below I_{final} , the boolean output discharging gets true after t_c . When V_{min} of any cell is reached during discharging, the boolean output charging gets true after t_d . Therefore it is possible to cycle a cell or a stack within its voltage limits V_{min} and V_{max}

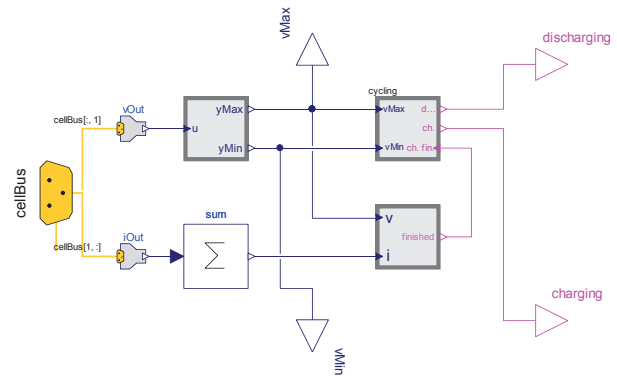


Figure 12: VoltageCycling model in the BatteryManagement package.

(in combination with a charging device and a load).

2.3 Chargers

In the Chargers package there is a constant current, constant voltage (CCCV) charging device modeled. A CCCV charging device charges a battery with a constant current until its CVL is reached (constant current phase). Then a constant voltage is applied and the current decreases (constant voltage phase). The charging device is switched off when the charging current reaches the final charging switch off current (typically 5% of the 1h discharging current).

When the boolean input of the modeled charging device is true (e.g. the boolean charging output charging from the VoltageCycling model described in section 2.2), a current source provides a constant current. This current is controlled by a voltage input (e.g. the maximal cell voltage in a battery stack), a reference voltage (parameter V_{max}) and limited by the maximal charging current (parameter I_{max}).

2.4 Loads

In the Loads package there are models to discharge a cell or a stack to cover typical lab situations (for example discharging a battery with a defined current or power profile). In extension to the electrical current sources (a current source parametrized with a negative current can be used as a load) there are four different loads considered:

- BooleanExternalControlledLoad
- BooleanConstantCurrent
- BooleanConstantPower

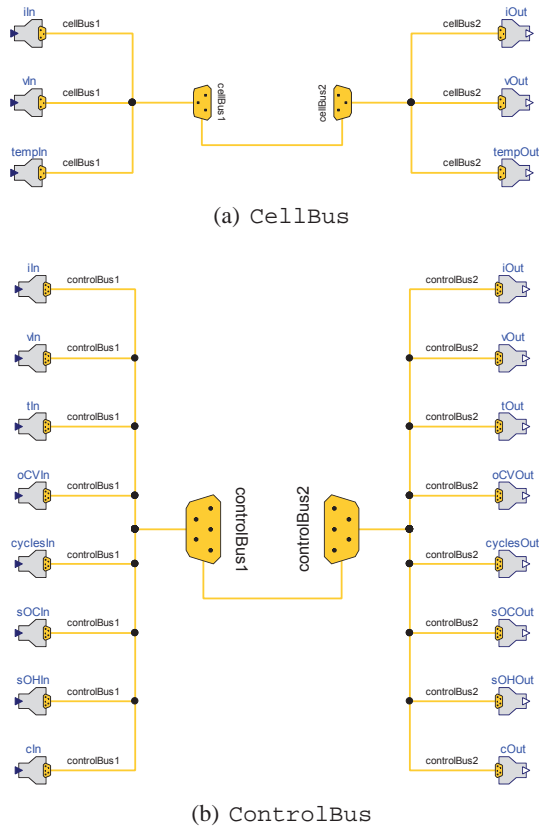


Figure 13: Structure of the CellBus (a) and the ControlBus (b) with bus adaptors to all variables on the bus (v , i , T for the CellBus and v , i , T , OCV , SOC , SOH , $cycles$, C for the ControlBus).

- SignalPower

All Boolean models have a boolean input on which for example can be connected to the boolean output discharging of the VoltageCycling model (cf. section 2.2). If on is false the cell or the stack is not being discharged.

2.5 Sensors

The Sensors package provides models to estimate the energy, the charge and the absolute charge from/to a cell or a stack. The models provided in the Electrical.Analog.Sensors package from the MSL are used and consequently extended.

2.6 Icons

In this package there are combined all icons used for the packages.



Figure 14: Test cell at constant temperature in a climate chamber.

2.7 Interfaces

There are two buses available: The CellBus and the ControlBus. The CellBus contains only measurable variables such as the cell voltage v , the cell current i and the cell temperature T . It can be used when lab situations are simulated and only measurable variables are significant. It can also be used to test different battery monitoring systems, based on i , v and T . The ControlBus contains the variables from the CellBus and additionally the following: OCV , SOC , SOH , $cycles$ and C . It could for example be the communication between the battery management system and the control units in an electric vehicle simulation.

In the Interfaces package there are also bus adaptors to extract/inject all these variables (Real values) from/to the CellBus as well as from/to the ControlBus. Figure 13 shows the structure of the CellBus and of the ControlBus with the usage of all bus adaptors.

3 Examples

Two different examples are presented:

First a LinearDynamicImpedance single cell model with one ($num = 1$) RC element is parameterized according to [9]. A realistic current profile, gained from the FTP72 cycle is continuously applied to the parameterized model as well as to the real cell [10]. The FTP72 cycle is a standardized real life driving cycle that simulates an urban route of 12.07 km.

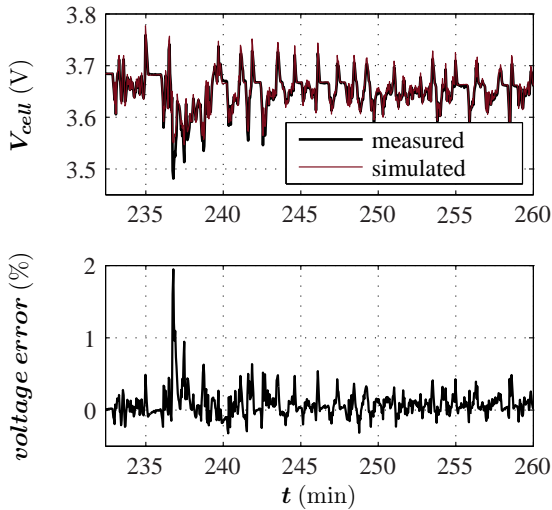


Figure 15: Comparison of the measured and simulated cell voltage when applying the FTP72 current profile to the real cell and to parameterized model (top). The error refers to the measured voltage (bottom).

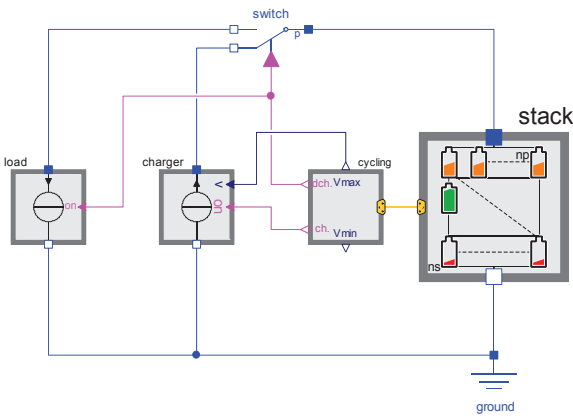


Figure 16: Simulation arrangement for cycling a battery stack.

This cycle is applied to a compact electric vehicle to extract the realistic current profile. The current profile is repeated starting from the full charged cell until it reaches discharging voltage limit DVL . This experiment is performed at constant temperature and Fig. 14 shows the test cell in a climate chamber. Fig. 15 compares the simulated to the measured cell voltage. The error, related to the measured voltage, between the simulated and the measured voltage stays below 2% at $SOC \approx 0.5$ during one complete FTP72 cycle. Therefore the chosen model approach is appropriate.

For the second example, three serially connected cells using the `LinearDynamicImpedance` stack model (without temperature and cell connectors) with different capacities are simulated. The cell capacities are $C_1 = 35$ Ah, $C_2 = 40$ Ah, $C_3 = 45$ Ah and

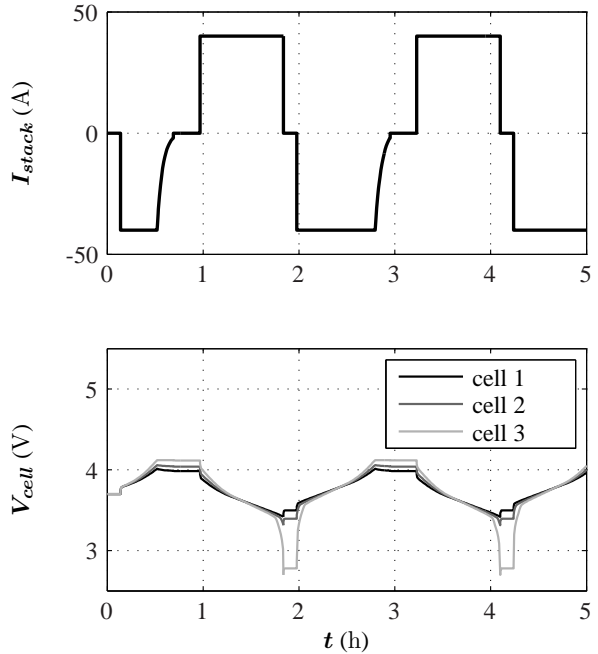


Figure 17: Current profile and voltage response from the three cells from the example simulation.

parametrized using [5, 9]. The cells are cycled with a 40 A charging and a 40 A discharging current between $CVL = 4.1$ V and $DVL = 2.7$ V with a delay after charging $t_c = 100$ s and after discharging $t_d = 50$ s. Cycling means continuously charging and discharging, which is a typical lab application for batteries. Fig. 16 shows the simulation arrangement with the charging device (charger), the battery management (cycling), the battery stack (stack), the discharger (load) and a switch (switch). Fig. 17 shows the battery stack current and the voltage from all three cells.

4 Conclusions

The Electric Energy Storage library, which is intended to be included in the Modelica Standard Library, has been described. The structure of the library and key parameters as well as the equations of the models for determining the output variables have been presented. Two typical lab tests for batteries have been simulated as an example to show the interaction of the models and to show the correctness of the model approach.

With the Electric Energy Storage library the user has a powerful tool to cover various applications of energy storages with simulation. In combination with existing libraries, (e.g. the SED) the behavior of electric energy storages and the interaction for example with the pow-

ertrain of an electric vehicle can be analyzed in detail. Future work will focus on including of thermal models of electric energy storages and the coupling with the electrical models.

References

- [1] J. V. Gragger, H. Giuliani, C. Kral, T. Bäuml, H. Kapeller, and F. Pirker, "The smartelectric-drives library - powerful models for fast simulations of electric drives," *Proceedings of the 5th International Modelica Conference*, vol. 2, pp. 571–577, 2006.
- [2] Modelica Association, "Modelica - a unified object-oriented language for physical systems modeling, language specification version 3.2," <http://www.Modelica.org/>, 2010.
- [3] Dassault Systems, "Dymola 7.4," <http://www.3ds.com/>, 2010.
- [4] P. Fritzson, *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*. IEEE Press, Wiley-Interscience, 2004.
- [5] *EIG ePLB CO20 lithium ion polymer cell datasheet*, EIG, 2010.
- [6] A. Jossen, "Fundamentals of battery dynamics," *Journal of Power Sources*, vol. 154, pp. 530–538, 2006.
- [7] S. Buller, "Impedance-based simulation models for energy storage devices in advanced automotive power systems," *Aachener Beitrage des ISEA*, vol. 31, 2002.
- [8] M. Einhorn and J. Fleig, "Improving of active cell balancing by equalizing the cell energy instead of the cell voltage," *Proceedings of the 25th International Battery, Hybrid and Fuel Cell Electric Vehicle Symposium and Exposition (EVS25)*, 2010.
- [9] M. Einhorn, F. V. Conte, C. Kral, J. Fleig, and R. Permann, "Parametrization of an electrical battery model for dynamic system simulation in electric vehicles," *Proceedings of the IEEE Vehicle Power and Propulsion Conference (VPPC)*, September 2010.
- [10] *FTP72 Urban Dynamometer Driving Schedule (UDDS)*, U.S. Environmental Protection Agency.

Implementation of a transmission line model for fast simulation of fluid flow dynamics

Stéphane Velut Hubertus Tummescheit
Modelon AB, Ideon Science Park, 22370 Lund, Sweden

Abstract

An implementation of a lumped and 1-dimensional pipeline model for simulation of fast pressure and flow transients such as water-hammer effects is presented. It is an extension of the classical Transmission Line Model (TLM), a transfer matrix representation of a pipeline, relating pressure and volume flow rates at the extremities of a pipeline. The proposed model has extended previous work in different aspects. The extensions were developed for the detailed operational investigation of a pipeline for the transport of carbon dioxide from a carbon capture plant to a suitable location for the geological storage of supercritical, dense phase carbon dioxide. A lumped temperature model, derived as the TLM model by integrating the distributed dynamics, has been added to describe the effect of heat losses in long pipelines. A dynamic friction model that is explicit in the medium and pipeline characteristics has also been included. Finally, it is shown that, with simple adjustments, the model can reasonably well describe the pressure dynamics in turbulent flow conditions. Some simulations have been carried out to compare the performance of the proposed model to the one from the Modelica Standard Library, and the results were also compared to measurement results from the literature. The resulting model has become useful for a wide variety of engineering applications: pipelines for gas and oil, district heating networks, water distribution networks, wastewater systems, hydro power plants and more. In the lumped, constant temperature version, there are no discretization artifacts, and even in the discretized version taking into account spatial and temporal changes in temperature, discretization artifacts are much smaller than for the standard finite volume model. Moreover, the short simulation times make the model suitable for real-time applications.

Keywords: water-hammer; transmission line model; dynamic friction; lumped model, CO2 transport

1 Introduction

A pipeline is a distributed system and its dynamics is described by partial differential equations. Most of the methods to compute flow transients are based on a spatial discretization of the pipeline into small segments. For accurate simulations of long ducts, a high discretization level is necessary, which leads to time-consuming computations. When only pressure and flow at the extremities are of interest, it is possible to capture the pipeline dynamics with a low order lumped model. The paper presents the implementation of such a model, the Transmission Line Model, based on the work presented in [5]. The original model has been extended to include temperature dynamics, an improved dynamic friction model, the effect of static head and some handling of turbulent flow conditions. The model presented in this paper captures the oscillations of the wave equation accurately with a lumped model, under the assumption of constant or slowly varying fluid properties. For suitable assumptions, the model is both more accurate with respect to measurements and has a much faster execution time than discretized models. It is suitable for simulation of larger multi-domain systems with variable time-step solvers, and also for pipeline systems with long (hundreds of kilometers) pipes. In contrast to discretized models that include momentum dynamics, the TLM model works fast and reliably also for zero-flow conditions.

The model is used in Modelon's Hydraulics Library as the long-line model without the thermal effects, and in Modelon's CombiPlant Library including the thermal effects.

2 Background

2.1 Fundamental equations

One dimensional pressure and flow dynamics in a circular pipeline is described at low Mach numbers

($q/A \ll c$) by coupled partial differential equations:

$$\begin{aligned} \frac{\partial p}{\partial t} + \frac{\rho c^2}{A} \frac{\partial q}{\partial x} &= 0 \\ \frac{\partial q}{\partial t} + \frac{A}{\rho} \frac{\partial p}{\partial x} + f(q) &= 0 \end{aligned} \quad (1)$$

where q is the averaged flow rate at any section, p the pressure, ρ the medium density, c the speed of sound and A the cross-section area. The friction factor $f(q)$, which describes the pressure losses per unit of length is defined by:

$$f(q) = \frac{\tau_{wall}}{0.5 \times \rho u^2} \quad (2)$$

where $u = q/A$ is the fluid velocity and the wall shear stress τ_{wall} is related to the velocity gradient in the radial direction:

$$\tau_{wall} = \mu \frac{\partial u}{\partial y} \quad (3)$$

and μ is the dynamic viscosity of the fluid.

2.2 Friction model

The friction factor, which depends on the medium viscosity and on the flow regime may be experimentally derived or in some cases analytically computed on physical considerations. In the case of steady state flow, $f(q)$ is related to the Fanning friction factor $\lambda(q)$ by

$$f_s(q) = \lambda(q) \frac{\pi D u^2}{8} \quad (4)$$

When the flow is laminar, the Darcy-Weisbach equation $\lambda(q) = \frac{64}{Re}$, Re being the Reynolds number, leads to

$$f_s(q) = \frac{32\nu}{D^2} q \quad (5)$$

and the equations (1) are therefore linear. In the case of turbulent flow, the Fanning friction factor can be described by the Colebrook-White equation and the resulting equations (1) are not linear.

It is commonly assumed that the steady state relations for f are also valid dynamically, which is actually not the case during fast transients such as water-hammers. Indeed, under fast transients, the average flow is influenced by 2D effects that the static friction model does not capture well. In [8], the author derived a dynamic wall shear stress model $\tau_{wall}(q)$ by solving

analytically the two dimensional Navier-Stokes equations for laminar flow, which resulted in the following expression for f :

$$\begin{aligned} f(u) &= f_s(u) + f_u(u) \\ &= \frac{8\rho\nu}{R^2} u + \frac{\rho\nu}{R^2} \int W(t-\tau) \frac{\partial u}{\partial t} d\tau \end{aligned} \quad (6)$$

where the unsteady term f_u depends on the weighting function W with an analytical but irrational expression in the frequency domain. It was approximated in the time-domain for transient simulations with the method of characteristics. Zielke's approach was later extended to derive a dynamic friction model valid in turbulent regime, for smooth pipes [7] and rough pipes [2]. The resulting model is as in the laminar case described by an irrational transfer function, which is in that case also dependent on the Reynolds number.

2.3 Implementation

Time-domain methods

Fluid flow transients in a pipeline may be simulated using equation (1) together with a friction description and boundary conditions. Powerful commercial solvers implementing various Computational Flow Dynamics techniques such as Finite Element or Finite Volume methods, are available to numerically solve those equations. Most of the techniques are based on a spatial discretization of the pipeline into small segments. For an accurate result in case of long pipelines, many segments are required, resulting in long computation times. Another limitation is the difficulty to simulate multi-domain models with complex time-varying boundary conditions.

A simpler and well-established technique for fluid flow simulation is the method of characteristics (MOC), which transforms the PDE (1) into two sets of ODE that can successfully be solved by fixed-step solvers. The fixed connection between the spatial and the temporal discretizations is the main limitation of the MOC technique: it results in many segments when the pipeline is long and it cannot be connected to variable step-solvers.

Frequency-domain methods

The second type of approach is based on a representation of the pipeline in the frequency domain. This is a well-suited method for real-time simulations and for system analysis such as control design or dynamic optimization. The fundamental equations (1) are first

Laplace-transformed and thereafter analytically integrated over the pipe length L :

$$\begin{pmatrix} P(s) \\ Q(s) \end{pmatrix}_{x=L} = \begin{pmatrix} \cosh\Gamma(s)L & -Z_c(s)\sinh\Gamma(s)L \\ -\frac{\sinh\Gamma(s)L}{Z_c(s)} & \cosh\Gamma(s)L \end{pmatrix} \begin{pmatrix} P(s) \\ Q(s) \end{pmatrix}_{x=0} \quad (7)$$

The equations involve non-rational terms and cannot be efficiently implemented without approximation. Two approaches are found in literature: the modal description and the Transmission Line approach. In the modal description, the irrational functions are approximated as truncated sum of low order linear filters whereas the TLM approach makes use of both linear filters and time-delays. As the TLM model describes explicitly the inherent delay of the wave propagation it results in lower orders model than the modal description.

The pipeline model that is described in the current paper is based on [5]. A block diagram of the pipeline model is shown in Figure 1. Every block in the schematic representation has a well-defined interpretation:

- Z_c is the line impedance describing the immediate and local effect of a flow change on pressure. It is modelled by a static gain $\frac{\rho c}{\pi(D/2)^2}$
- $R(s)$ is the hydraulic resistance of the duct and determines the pressure drop to the flow at stationarity. It was described by $R(s) = \frac{R_0}{\kappa T s + 1}$ where $R_0 = \frac{\Delta p_\infty}{q_\infty}$
- $e^{-sT} = e^{-s\frac{L}{c}}$ is the delay associated to the time it takes for a pressure wave to travel through the pipeline at the speed of sound c .
- $G_f(s) = G_f^1(s)G_f^2(s)$ is a dynamic filter that models the attenuation of the pressure disturbance when the wave goes from one extremity to the other. $G_f^1 = \frac{s/\omega_2 + 1}{s/\omega_1 + 1}$ and $G_f^2(s)$ describe the effect of static and dynamic frictions, respectively. The frequencies ω_1 and ω_2 are given by $\omega_1 = c/(\kappa L)$ and $\omega_2 = \omega_1 e^{-\frac{R_0}{2Z_c}}$. G_f^2 needs to be optimized for every medium and pipeline characteristics.

3 A novel lumped pipeline model

The transmission line model [5] has been implemented in Modelica and further developed to describe the following characteristics:

- heat loss. When heat loss cannot be neglected, temperature is not constant throughout

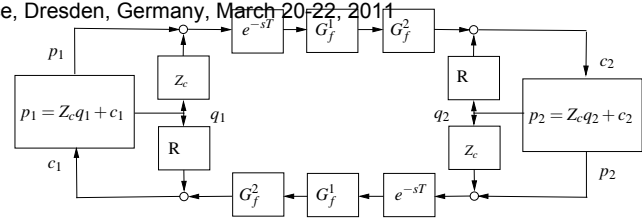


Figure 1: Schematic representation of the Transmission Line Model. Z_c is the line impedance, R is the hydraulic resistance, e^{-sT} is a time-delay and G_f^1, G_f^2 describes the effect on static and dynamic frictions on the travelling pressure waves.

the pipeline. The temperature profile and its influence on the pressure wave dynamics need to be included in the TLM model.

- static head, in the case of non-horizontal pipelines.
- an improved description of the dynamic friction
- turbulent flow

3.1 Turbulent flow condition

Linearity of the continuity and momentum equation is an essential assumption in the derivation of the transfer matrix representation. The turbulent flow regime introduces a nonlinearity in the friction term f and the coupled PDEs (1) can no longer be integrated explicitly. It is still possible to consider small deviations from an equilibrium point and linearize the equations (1) around a stationary point:

$$\begin{aligned} \frac{\partial \Delta p}{\partial t} + \frac{\rho c^2}{A} \frac{\partial \Delta q}{\partial x} &= 0 \\ \frac{\partial \Delta q}{\partial t} + \frac{A}{\rho} \frac{\partial \Delta p}{\partial x} + \frac{\partial f}{\partial q} &= 0 \end{aligned} \quad (8)$$

Moderate pressure and flow deviations from a stationary point can therefore be simulated by typically changing the hydraulic resistance R_0 in the TLM model with the linearized resistance R_l , which is a parameter used in both $R(s)$ and $G_f^1(s)$:

$$R_l = \frac{\partial p}{\partial q} = \frac{\rho L}{A} \frac{\partial f}{\partial q_0} \quad (9)$$

To get correct stationary pressure drops under larger pressure or flow changes, the parameter R_0 in $R(s)$ has not been linearized and $R_0 = \frac{\rho L}{q_0 A} f(q_0)$ has instead been used. Note that the changes for handling turbulent flows do not affect the original model in the laminar flow regime.

3.2 Dynamic friction

In the original TLM model from [5], the transfer function G_f^2 —describing the frequency dependent friction—needs to be tuned for the considered pipe and medium. To avoid this optimization, the friction model from [4] that is explicit in both the medium properties and the pipe characteristics has been implemented. In this model, the transfer function G_f^2 is expressed as a sum of k linear filters, approximating the analytical solution:

$$G_f^2(s) = 1 - \frac{8\nu L}{cD^2} \sum_{i=1}^k \frac{m_i \alpha s}{n_i + \alpha s} \quad (10)$$

where $\alpha = 4D^2/\nu$ and m_i and n_i are medium- and pipe-independent constants that affects the frequency range in which the approximation should be most accurate. Compared to [5] better agreement with experimental data has also been reported.

The friction description from [4] is theoretically valid only for laminar flow. In [1], it was found that the laminar flow approximation of the dynamic friction model is a reasonable basis for transient turbulent friction as long as the Reynolds number is moderate (below 10^5). The advantage of this model compared to [2] or [7] is that the dynamic friction model does not depend on the Reynolds number.

3.3 Static head

The basic pipeline model can easily be extended to account for pressure variations due to height differences. The equations at the pipe nodes are updated as follows

$$P_1 = Z_c Q_1 + C_1 - \rho g L \quad (11)$$

$$P_2 = Z_c Q_2 + C_2 + \rho g L \quad (12)$$

where ρ is the density of the medium at the pipe inlet, g is the gravity constant and L is the length of the pipe segment. In the previous equations it is assumed that the pipe node 1 is at a higher altitude than the pipe node 2.

3.4 Time-varying properties

Need for discretization

In the TLM model, it has been assumed that temperature is both constant in time and space. When heat loss along the duct is not negligible, this hypothesis is not valid and the variations in the medium properties cannot always be neglected. This is for instance true when the duct is very long or poorly isolated and when

the mass flow is time-varying or low. The pipeline needs somehow to be discretized into segments, each segment being characterized by uniform but possibly time-varying medium properties. The variations in the medium properties are however slow because the medium properties depend to a larger degree on temperature than pressure and temperature dynamics is slow.

To incorporate the temperature dynamics into the pipeline model, a similar approach as in the TLM model derivation has been used: the energy balance is explicitly integrated along the pipeline to get a relationship between the inlet and outlet temperature of every segment. In that way, the pipeline can be discretized into a moderate number of segments, each segment being the combination of a TLM model and a temperature dynamic. The segment length is typically much longer than in the MOC description and it is related to the amplitude of the temperature change along the line as well as to the sensitivity of the medium properties to temperature changes.

Lumped temperature dynamics

The one dimensional energy balance in the pipeline is represented by the following partial differential equation:

$$\frac{\partial T}{\partial t} + \frac{\dot{m}}{\pi R^2 \rho} \frac{\partial T}{\partial x} + \frac{1}{\pi R^2 \rho c_p} q(T(x)) = 0 \quad (13)$$

where $q(T(x))$ describes the heat loss to the surroundings along the line and depends on the boundary conditions. The heat loss may often be described as a linear function of the temperature difference between boundaries:

$$q(T(x)) = kS(T(x) - T_{boundary}) \quad (14)$$

where k is the heat conductivity of the surroundings (water, soil) and S is the shape factor. The shape factor S may take the following form [3]:

$$S = \frac{2\pi}{\operatorname{acosh} \frac{2h}{D}} \quad (15)$$

for a pipe with diameter D , buried h meters below the ground surface at constant temperature, or

$$S = \frac{2\pi}{\ln \frac{D_\infty}{D}} \quad (16)$$

for a pipe with diameter D and surroundings at constant temperature, a distance $D_\infty/2$ away from the pipe center.

The equations (13) and (14) are linear and can be integrated explicitly together with the boundary condition from the inlet temperature:

$$T(x = 0, t) = T_{in}(t) \quad (17)$$

When the mass flow is constant, it is sufficient to Laplace-transform (13) and integrate over the pipe length. An explicit solution to (13) can also be derived in the case of time-varying flows by applying the method of characteristics. Temperature at the outlet of a pipe with length L is then given by:

$$T(L, t) = T_{boundary} + (T_{in}(t - \tau) - T_{boundary})e^{-\frac{\tau}{T_p}} \quad (18)$$

The time-varying delay τ and the characteristic decay-time T_p are given by

$$\pi R^2 L = \int_{t-\tau(t)}^t \frac{\dot{m}(s)}{\rho} ds \quad (19)$$

$$T_p = \frac{c_p \rho \pi R^2}{kS} \quad (20)$$

3.5 Modelica implementation

The proposed pipeline model has been implemented in Modelica using the Dymola software. The basic pipeline segment, characterized by constant medium properties is composed of two models in parallel:

- a dynamic model describing pressure and flow dynamics, see Section 3.1 to 3.3
- a dynamic model describing temperature dynamics, see Section 3.4

The implementation of the pressure wave dynamics is done using linear transfer function blocks as shown in Figure 2. The implementation of the time-delays (associated with the wave propagation or the transport delay) is based on the external C-function called "transportFunction" available in Dymola. The functional equivalent of that function is currently considered in the Modelica Association to be included as operator `spatialDistribution()` into the Modelica language. This function allows to model delays based on a physical transport mechanism, like flow in a pipe, also in the case of bi-directional flow. Computation of the time-varying delay τ in the temperature dynamic is performed by using the differential form of Equation (19):

$$\frac{d\tau}{dt} = 1 - \frac{v(t)}{v(t - \tau)} \quad (21)$$

where $v = \frac{\dot{m}}{\rho}$ is the fluid velocity.

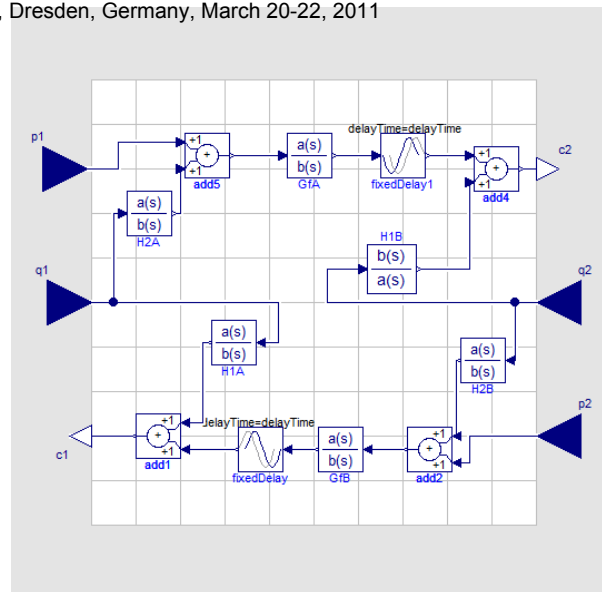


Figure 2: Implementation of the mass and momentum dynamics in Dymola.

4 Simulation

The original TLM model has shown very good agreement with experimental data and other pipeline models when the flow is laminar, see for instance [5] or [4].

4.1 Evaluation of the proposed model

Turbulent flow conditions

Simulations have been performed in Dymola to show that the proposed model gives reasonable results even in the case of turbulent flow and large pressure disturbances. The considered medium is water and the pipeline is characterized by a length of 1000 meters, a diameter of 0.035 meter and a relative roughness of 0.005. The pipe inlet is connected to an ideal flow source while its outlet is connected to an ideal pressure source ($p=20$ bar). Pressure waves are generated by fast variations of the inlet flow.

The TLM model is compared to an instance of the pipeline model from Modelica Standard Library with 50 segments. In the MSL pipeline model, the partial differential equations are treated with the finite-volume method and a staggered grid scheme for momentum balances. The dynamic friction model was not included in the simulation because it is not implemented in MSL. Note that such a friction model would give between 50 and 150 additional states in the MSL model (between 1 and 3 extra states per segment). Simulation results are shown in Figure 3. The pipeline is initially at rest and the flow at the inlet is

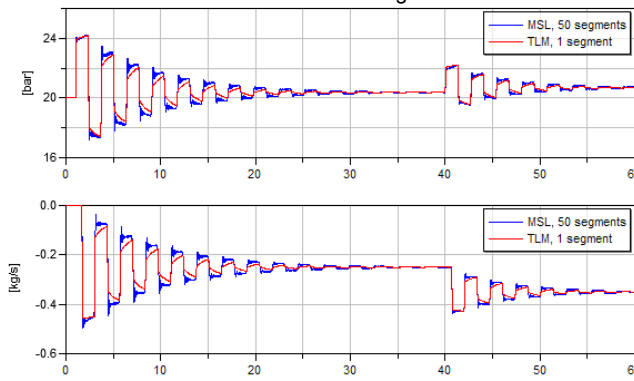


Figure 3: Comparison of the MSL and TLM pipeline models. Top: pressure at the inlet, bottom: mass flow rate at the outlet. The flow at the inlet is changed at $t=1$ s and $t=40$ s.

varied at two time instants. The first flow increase, at $t = 1$ s, generates pressure waves of relatively large amplitude and moves the fluid in a very short time to the turbulent regime ($Re \approx 10^4$). Despite this fast and large transients into the turbulent regime, the performance of the TLM model is good:

- As expected, the frequency of oscillations is very good.
- The amplitude and the shape of the first peak is very good.
- The overall attenuation rate of the travelling wave is good. One can notice a slightly higher damping with the TLM model.
- The noisy signals in the MSL model due to the discretization artifacts are replaced with a smooth signal.
- The simulation time is much shorter with the TLM model: 8.6 instead of 100 seconds.

The second perturbation is smaller and results therefore in a better agreement between the models. A slightly higher damping can again be observed with the TLM duct model. To investigate whether the difference is mainly caused by the model structure or by the parameter values, the static friction term G_f^1 has been adjusted to give a slightly lower damping. For that sake, the frequency w_2 has been changed from $w_1 e^{-\frac{R_l}{2Z_c}}$ to $w_1 e^{-\frac{R_l}{3Z_c}}$. The results shown in Figure 4 are much better and confirms that the model structure may be suitable for turbulent flow simulations. Further analysis and simulations are however required to fully validate the model and to eventually derive a Reynolds

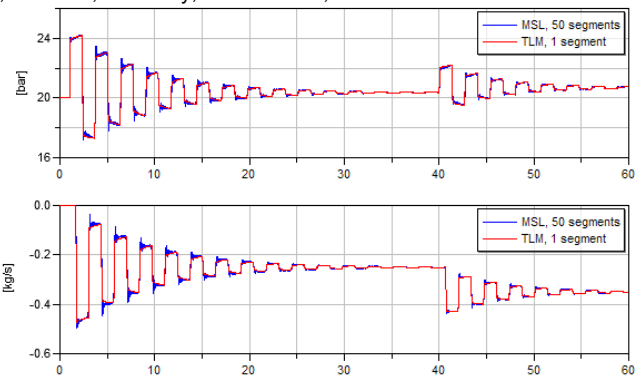


Figure 4: Comparison of the MSL and a slightly modified TLM pipeline models. Top: pressure at the inlet, bottom: mass flow rate at the outlet. The flow is changed at $t=1$ s and $t=40$ s.

dependent parametrization of the transfer function G_f^1 .

Temperature dynamics

The reference model is again the pipeline model from MSL and the pipeline characteristics are identical to the ones given in previous section. Concerning the heat transfer, an ideal heat transfer described by a coefficient $\alpha = 10 \text{ W/K/m}^2$ has been chosen and the pipeline surroundings have been assumed to be at constant temperature $T_{boundary} = 10^\circ\text{C}$. The effect of changes in both the mass flow rate and the inlet temperature on the outlet temperature are investigated.

Simulation results are shown in Figure 5. The initial state is characterized by a mass flow rate of 0.25 kg/s and an inlet temperature of 23.4°C . The resulting outlet temperature at steady state is about 14.8°C with both models. At time $t=0.25$ h, the inlet temperature is linearly decreased to 1.5°C . The dynamic response of the TLM model differs substantially from the MSL model. In the MSL case, the outlet temperature starts decreasing before the cold water at the pipe inlet has been transported to the outlet. This is due to the spatial discretization of the pipeline model, which is equivalent to a mixing effect. The TLM model, implemented with a pure delay operator, does not present this mixing property and captures well the effect of the transport delay. When the number of nodes is increased the response of the MSL model tends towards the TLM solution, but at the cost of a longer simulation time. At time $t=2$ h, the mass flow rate is decreased to 0.1 kg/s. It has a slow but immediate effect on the outlet temperature. The response of both models are compa-

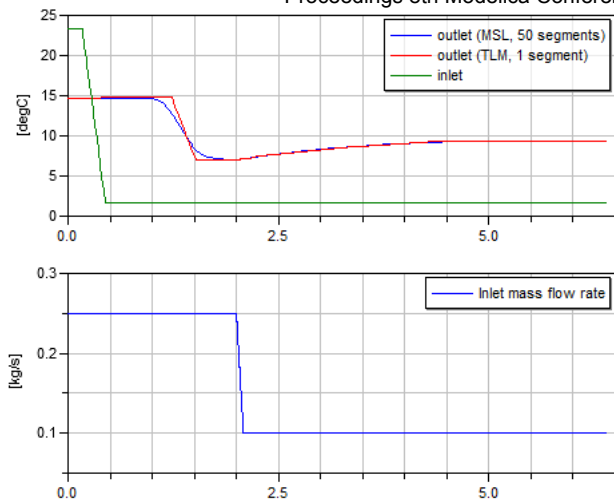


Figure 5: Temperature dynamics of the MSL and TLM pipeline models. Top: outlet and inlet temperatures. Bottom: inlet mass flow rate.

rable.

The simulation times are approximately 0.9 second for the TLM model and 63.0 seconds for the MSL model.

4.2 Application: transport of supercritical carbon dioxide

Successful implementation of the CO_2 capture and storage techniques is largely dependent on the success with which CO_2 can be economically and safely transported from the power plants to the storage sites. As safety is of paramount importance, any risks that may prevent the safe operation of CO_2 transport pipelines must be identified and subsequently eliminated or controlled. One of the risks is associated with the formation of gas phase CO_2 within the pipeline resulting from a decrease in pressure or increase in temperature. Two phase flow can lead to the occurrence of cavitation or water-hammer with the associated problems of noise, vibration and pipe erosion and ultimately, pipe failure.

The pipeline model presented in the current paper has been used to investigate how the physical state of CO_2 is affected during normal and failure modes such as quick shut-down, compressor stop or load changes, see [6].

5 Conclusion

A lumped pipeline model for fast simulation of pressure and flow transients in pipelines has been pre-

sented. It is an extension of the classical Transmission Line Model, a transfer matrix representation of a pipeline characterized by constant medium properties and laminar flow conditions. The proposed model has extended the basic TLM model to describe the influence of heat losses. A dynamic friction model that is explicit in the medium and pipeline characteristics has also been included. Finally, it is shown that, with simple adjustments, the model can reasonably well describe the pressure dynamics in turbulent flow conditions. Some simulations have been carried out to compare the performance of the proposed model to the one from the Modelica Standard Library. It turns out that the model accuracy is satisfactory and that the short simulation time makes it suitable for real-time applications. The model has also been applied to simulate different operation modes in a CO_2 transfer pipeline.

References

- [1] H. Kuo-Lun A. E. Vardy, J. M. B. Brown, *A weighting function model of transient turbulent pipe friction*, Journal of Hydraulic research **31** (1993), no. 4.
- [2] J. M. B. Brown A. E. Vardy, *Transient turbulent friction in smooth pipe flows*, Journal of Sound and Vibration **259** (2003), no. 5, 1011 – 1036.
- [3] T. L. Bergman-A. S. Lavine F. P. Incropera, D. P. Dewitt, *Introduction to heat transfer*, John Wiley & Sons, 2007.
- [4] D. N. Johnston, *Efficient methods for numerical modeling of laminar friction in fluid lines*, Journal of Dynamical Systems, Measurement and Control **128** (2006).
- [5] S. Gunnarsson P. Krus, *Distributed simulation of hydromechanical systems*, Third Bath International Fluid Power Workshop.
- [6] M. T. P. Mc Cann-H. Tummescheit S. Velut S. Liljemark, K. Arvidsson, *Dynamic simulation of a carbon dioxide transfer pipeline for analysis of normal operation and failure modes*, 10th International Conference on Greenhouse Gas Technologies, 2010.
- [7] A. E. Vardy and J. M. B. Brown, *Transient turbulent friction in fully rough pipe flows*, Journal of Sound and Vibration **270** (2004), no. 1-2, 233 – 257.

- [8] W. Zielke, *Frequency dependent friction in transient pipe flow*, Ph.D. thesis, University of Michigan, 1966.

Fluid Simulation and Optimization using Open Source Tools

Kilian Link, Stephanie Vogel
Siemens AG

kilian.link@siemens.com, vogel.stephanie@siemens.com, ines.mynttinen@tu-ilmenau.de

Ines Mynttinen
Technical University Ilmenau

Abstract

Many open-source (OS) tools exist in the Modelica universe. OS tools have the benefit that they are freely available and fit well to Modelica as a non-proprietary language. However, the industrial usage of these tools seems to be very limited so far. Despite this fact, fluid modeling is possible if restrictions are taken into account.

In this contribution, we propose to use benchmark models for the systematic investigation of the accuracy and the performance of Modelica OS tools. The present implementation circumvents the limitations of OpenModelica making it possible to simulate the model system despite existing restrictions. Beside simulation tasks Modelica-based optimization is possible using the OS tool JModelica. However, care must be taken with respect to the model features. In particular instantaneous transitions within the system dynamics, such as phase transitions, switching of valves with discrete behaviour or flow reversal represent a severe obstacle for optimization. In this article, we present a parameter estimation problem including instantaneous changes of the flow direction. In addition, an example of model predictive control (MPC) for a control task difficult to solve with conventional methods is shown.

Keywords: Fluid Simulation; Optimization; OS Modelica Tools

1 Introduction

Modelica is the preferred modeling language for dynamic simulations within Siemens Energy [1] due to its applicability for multi-domain modeling of physical systems and the high degree of maintainability of Modelica models. The Modelica Libraries Modelica.Media and Modelica.Fluid provide basic elements to model pipe networks including, e.g., economizers, super heaters and evaporators which are essential parts of each power plant. However the flexible approach of Modelica.Fluid makes it unsuitable for daily business in a well defined application

area. Thus well proven models of these components exist in the in-house library SiemensPower.

The commercial tool Dymola is used for modeling and simulation. The alternative tool OpenModelica [2] is an OS Modelica-based modeling and simulation environment intended for industrial and academic usage, which has the large benefit that it is freely available and fit well to Modelica as a non-proprietary language. However, the tool support for fluid modeling is limited due to some advanced Modelica features, e.g. the usage of Modelica.Media and Modelica.Fluid. Despite this fact, fluid modeling is possible if the functions missing in OpenModelica are called from external libraries. In order to measure the quality of OS Modelica tools compared to the established commercial software, e.g. Dymola with respect to the accuracy and the performance benchmark models are needed. In this way the systematic investigation of models with increasing size and complexity can reveal bottlenecks and shortcomings in OpenModelica. To go beyond simulation applications towards optimization, the Modelica-based open source platform for optimization, simulation and analysis of complex dynamic systems JModelica [3] is the most preferable choice. The main objective of the project is to create an industrially viable open source platform for optimization of Modelica models. The three-level structure of the user interface is probably its main advantage, since it allows for convenient implementation of user specified applications. The issues of compilation, data processing, setting up the algorithm and starting the optimization are well addressed in the Python script file. Furthermore, the Python script file serves to store the data and to do some customized plotting. These capabilities for Python scripting considerably reduce the effort to implement user applications. By means of the Optimica extension to Modelica, the optimization problem itself is formulated at the middle level implementing the objective function and the constraints using the special class optimization. At the lowest level of the JModelica user interface the dynamic model is defined. We used JModelica for solving the parameter estimation problem of a hybrid dynamic system as well as an off-line model

predictive control (MPC) problem presented in sections 3 and 4, respectively.

2 Simulation of Fluid Models with OpenModelica

Fluid Modeling based on OpenModelica is not very common so far. This is mainly due to the following restrictions still present in OpenModelica. As mentioned above, the support of Modelica.Fluid and Modelica.Media is limited. On the other hand, the performance of the solver (the so called back-end) is very poor, i.e. solving real life problems is not possible up to now. However enormous improvements are on the way supported by joint efforts in the OPENPROD [11] project. In order to evaluate the improvements, the availability of benchmarks and realistic test cases is a natural and essential first need. The size and complexity of these benchmark models should be easily adaptable. Furthermore, the model is desired to be valid in different phase regions, since phase transitions are crucial in fluid dynamics. In addition, it is useful to build up the model from a set of components, which are also well established in real technical systems to facilitate the extension of the model from a sandbox example to real world application.

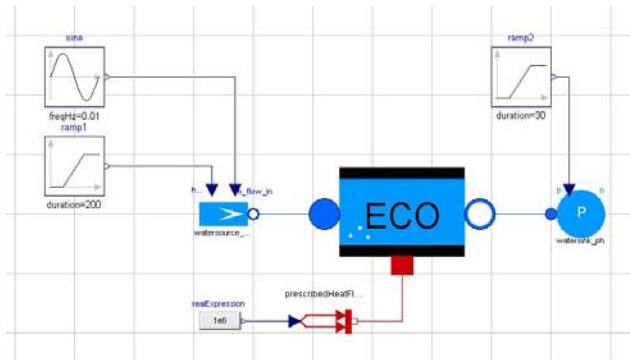


Figure 1: Heated pipe model

Figure 1 shows a model which can be used as such a benchmark. The central part of the model is a heated pipe which is connected to a water source and a heating element. The liquid channel of the pipe is discretized along the flow direction. The connections between these elements are represented by nodes.

The left hand node is connected to the water source which supplies the liquid flow. The heated metal wall of the pipe is modeled as cylindrical tube with the number of layers L in radial direction. The constant heat flow is distributed equally over all axial elements.

This quite simple model is very well suited as a benchmark model for the following reasons:

- The problem size can be easily adapted by changing the parameters for spatial discretization, see Table 1.
- Dynamic changes in the mass flow, enthalpy and pressure boundaries can be utilized to run through different phase regions.
- Based on basic components (i.e. tube plus boundaries) more complex models could be set up easily.

Discretization Parameters: N – number of nodes in flow direction L – number of tube wall layers	Continuous states
N = 3, L = 3	13
N = 10, L = 3	41
N = 30, L = 3	121
N = 200, L = 3	801
N = 100, L = 6	701

Table 1: Number of states depending on the spatial discretization of the heated pipe model

Although this model is very simple, some features of Modelica.Media which are not yet implemented in OpenModelica 1.6 are needed. To circumvent this problem, the model is rewritten such that direct functions compute all water properties without the use of the Media package of the Modelica Standard Library. The necessary water-steam functions have been substituted by external function calls of the TTSE (Tabular Taylor Series Expansion) [4] library. The TTSE uses a table of stored water properties and derivatives calculated with IAPWS (International Association for the Properties of Water and Steam) with pressure and enthalpy (p, h) or density and enthalpy (ρ, h) as variables. On each cell, the thermodynamic properties of water and steam are computed using the Taylor series expansion. In this way, TTSE offers fast computation with an acceptable accuracy for dynamic simulation. For the first investigation a small model corresponding to the first row in Table 1 has been used. The results from the well proven modeling environment Dymola in version 7.4 have been compared to those from OpenModelica version 1.6. In both tools 'dassl' with a tolerance of $1e-6$ was used.

The comparison of the results shows that they are nearly identical. This suggests that the solver seems to solve an identical problem in both cases. However, the differences in performance are huge. While Dymola solves the problem in some milliseconds, OpenModelica spends several minutes to solve the

problem. Facing that enormous difference running larger models does not make much sense.

Primary investigations are carried out to identify the reasons for the poor performance of OpenModelica 1.6.0.

The pipe model was simulated with an increasing number of continuous states depending on the nodes in flow direction. Different simulation options were used to identify the reasons for the poor performance.

OpenModelica 1.6.0 uses ‘dassl’ as standard solver. Since ‘dassl’ is so slow, ‘dassl2’ has been tried. This solver is considerably faster than ‘dassl’, but it still takes at least seven times as long as Dymola 7.4 using ‘dassl’. Table 2 shows the simulation time depending on simulation method. For higher discretization in the flow direction ‘dassl2’ is at least 100 times faster than ‘dassl’ generating the same results.

N (number of nodes)	solver	Simulation time
3	dassl	289.29 s
3	dassl2	2.52 s
10	dassl	803.81 s
10	dassl2	6.92 s
19	dassl	1523.63 s
19	dassl2	16.04 s

Table 2: Simulation time depending on solver and discretization

All test cases use the ‘plt’ output format, which is default and currently the only format capable of using plot functions. For testing it is better to use output format ‘bin’ to speed up calculation. For N>19 the compilation failed with an internal OpenModelica error. Probably the array sizes are too large.

We set up a special test case for external function calls to find out whether they cause considerable difference in the simulation time. We can exclude that the differences between Dymola 7.4 and OpenModelica 1.6.0 depend on external function calls, since the simulation time is quite similar for both simulation environments.

3 Parameter Estimation for a Hybrid System using JModelica

Parameter estimation is an important issue in many fields of industrial engineering [1], since it allows for efficient adaptation of system models.

Parameter estimation aims at extracting the best values of parameters determining the dynamics of the system under consideration, based on a series of measurements $x_{j\ell}^{(m)}$ of several state variables x_j , $j =$

$1, \dots, M$ at different time points t_ℓ , $\ell = 1, \dots, N$. Due to measurement error, the estimated parameters are subject to some uncertainty. Assuming that the measurement error is uncorrelated and normally distributed with variance σ_j^2 , model parameters can be estimated by minimizing the weighted least-squares function

$$J(\rho) = \sum_j^M \sum_l^N \frac{(x_j(t_l) - x_{j\ell}^{(m)})^2}{\sigma_j^2} \tag{1}$$

subject to the DAE system representing the system dynamics as equality constraints and variable boundaries as inequality constraints.

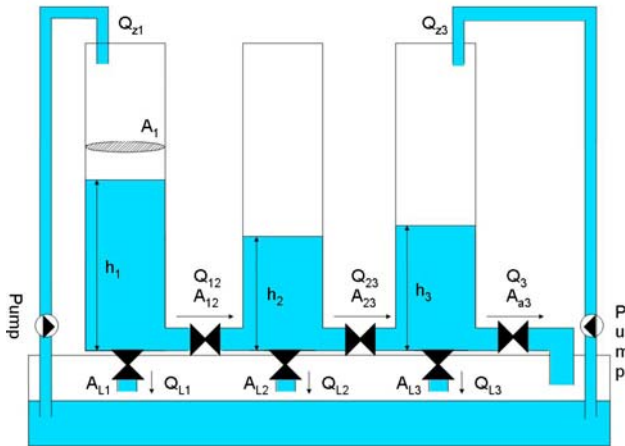
In this study we consider a parameter estimation problem for a hybrid system. Hybrid systems possess a mixed continuous and discrete behavior due to instantaneous mode transitions. In fluid systems the latter arises very frequently as a result of valves with discrete behaviour, phase transitions or flow reversal. Solving these kinds of optimization problems remains a challenging task and the use of available solvers is limited. The major difficulties lie in the discontinuous function values and gradients.

To overcome the difficulties we studied reformulation methods for hybrid systems. In these methods the so-called switching condition Ψ determines the value of a newly introduced continuous switching variable $\varphi(\Psi)$. In order to force this variable to meaningful values to guarantee at least an approximate instantaneous switch either a relaxation or a penalization method can be used. In this study we apply the Smooth Step Function (SSF) Approach with

$$(2) \varphi(\psi) = \frac{1}{1 + \exp(-\tau\psi)}$$

as a relaxation method with relaxation parameter τ and the Penalization of Incomplete Switching (PICS) as a penalization method [6]. In order to examine the capabilities of reformulation methods in parameter estimation for hybrid systems, we consider a tank system similar to those used in [7], [8], and [9].

The system consists of three tanks in a row connected to each other (Figure 2). There are inflows Q_{zi} , $i = 1, 3$ to the left and the right tank. The parameter estimation problem can be stated as:


Figure 2: Three-tank system

$$\min_{A_{12}, A_{23}} J(A_{12}, A_{23}) = \sum_j^N (h(t_j) - h_j^{(m)}) \quad (2a)$$

s.t.

$$A_1 \dot{h}_1 = Q_{z1} - Q_{12} - Q_{L1} \quad (2b)$$

$$A_2 \dot{h}_2 = Q_{12} - Q_{23} - Q_{L2}$$

$$A_3 \dot{h}_3 = Q_{z3} + Q_{23} - Q_{L3} - Q_3$$

$$Q_{ij} = A_{ij} \varphi_{ij} \sqrt{2g |\psi_{ij}|} \quad (2c)$$

$$\psi_{ij} = h_i - h_j, (i, j) = \{(1,2), (2,3)\}$$

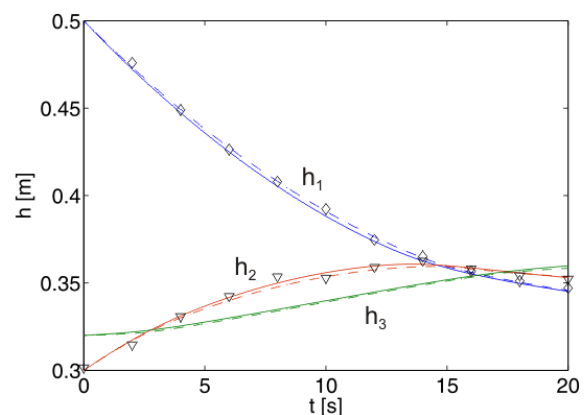
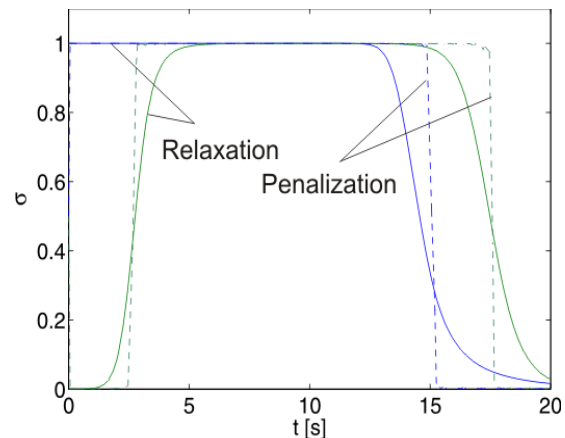
$$-1 \leq \varphi_{ij} \leq 1 \quad (2d)$$

The objective function (Eq. (2a)) is to be minimized subject to the dynamic model equations and bounds of the switching variable φ (Eq. (2b-d)). The dynamics of the tank levels h_i ($i = 1, 2, 3$) are given by the mass balance of the tanks (Eq. (2b)). The outflows Q_{Li} , Q_3 and the flows between the tanks are modeled by Torricelli's law (Eq. (2c)). In the original formulation, a sign function switches the direction of the flow between two tanks abruptly from +1 to -1 or vice versa, when the condition $\psi_{ij} = h_i - h_j = 0$ is passed. Since the gradient of the flow diverges to infinity at this point, in our reformulation the sign function in Eq. (2c) is replaced by the switching variable φ . The correct switching behavior should be ensured by the relaxation or penalization methods mentioned above.

We used JModelica for solving the nonlinear optimization problem. Here we exploit the three-level structure of JModelica's front end as described above. JModelica does not allow optimization problems including instantaneous mode transitions since in this case the gradients of the objective function or the constraints needed by the optimization algorithm are expected to be not well defined. However, the

relaxation and penalization methods used in this study lead to differentiable objective functions and constraints. The reformulation method can easily be implemented in JModelica.

Our aim is to estimate via minimization of the objective function (Eq. (2a)) the flow parameters A_{ij} based on (simulated) measurement data $h_\ell^{(m)}$, $\ell = 1 \dots 10$ of the tank levels taken equidistantly within the time horizon $(t_0, t_f) = (0, 20)$ s. The data are generated via simulation of the original model with added Gaussian noise. The optimal state trajectories found for the parameter estimation problem are shown in Figure 3).


Figure 3: State trajectories SSF (solid) and PICS (dashed) as well as the measurements of h_1 (diamonds) and h_2 (triangles)

Figure 4: The corresponding switching variables φ_{12} (blue) and φ_{23} (green)

They agree quite well with each other. In particular, the crossing points of the levels h_1 and h_2 and the levels h_2 and h_3 nearly coincide. This reflects the fact that the correct switching behavior is obtained in both cases as shown in Figure 4. It can be seen that using the relaxation method the switch is smooth and rather slow, which apparently has almost no impact on the trajectories. In contrast, when using the pe-

nalization approach the switch takes place almost instantaneously. Table 3 compares the optimal parameter values estimated by the two methods. With penalization we obtain a very good value for the parameter A_{12} , but the deviation of A_{23} from the exact value is considerable. This is mainly due to the fact that the objective function is much more sensitive to A_{12} than to A_{23} . The relaxation method results in moderate deviations for both parameters. In summary, both approaches provide reasonably accurate results.

	exact	Relaxation	Penalization
$A_{12} [10^{-5} \text{m}^2]$	6.0	6.51	6.06
$A_{23} [10^{-5} \text{m}^2]$	2.0	2.29	2.98

Table 3: Optimal parameter values

We also studied the ability of the SSF algorithm to cope with random errors in the data set, which is inevitable in real data acquisition. The variance of the measurement is varied in the range $\sigma_M = [0.5, 10] \cdot 10^{-4} \text{ m}$. The parameter estimation is carried out for 50 series of h_2 for each σ_M and the mean parameter values as well as their variance σ_p are achieved (see Figure 5).

It can be seen that the mean values of the parameter stay constant over a wide range of random errors. Obviously, a higher variance of measured data leads to a higher variance of the estimated parameter value. A strict proportionality of σ_M and σ_p is expected in the case of the measured quantity (here h_2) linearly depending on the parameter (here A_{12}).

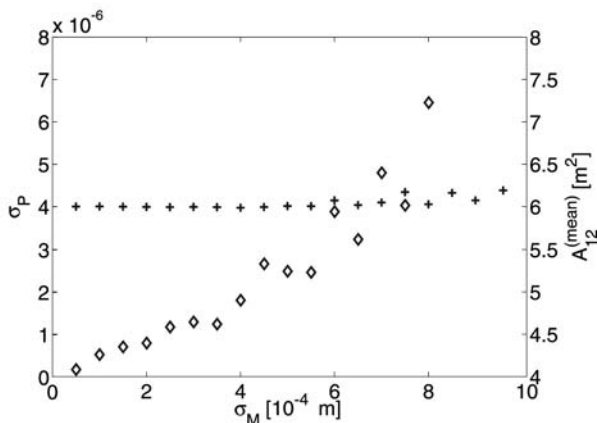


Figure 5: Mean parameter value A_{12} (crosses) and corresponding variance σ_p (diamonds) in dependence on the variance of measurement σ_M .

4 Nonlinear Model Predictive Control using JModelica

Nonlinear model predictive control (NMPC) is an advanced technique to solve challenging optimal control problems. In this method, the optimal control problem is formulated as constrained dynamic optimization problem, where the constraints are given by the dynamic model of the plant and the process restrictions. This problem is solved for the so-called prediction horizon T_p (see Figure 6). The resulting optimal controls within the so-called control horizon T_c are applied to the plant. At $t = T_c$, the time horizon of the optimal control problem is moved to this new initial point. The measurement of the present plant state serves as feedback. The initial conditions are accordingly updated and the problem is solved again. Since the optimal control problem is solved once per move of the time horizon, T_c is the CPU time available to solve the problem.

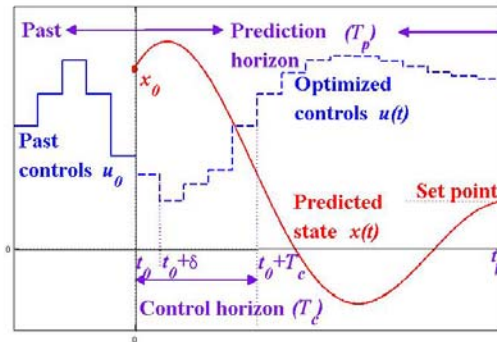


Figure 6: Principle of NMPC [9]

It is straight forward to carry out the NMPC based on power plant models primarily developed for dynamic simulation applications. The choice of test cases will naturally focus on optimal control problems which are difficult to solve by conventional controllers. Besides that, one has to take into account the size and complexity of the plant model as a limiting factor with respect to the computational effort.

In this study, the temperature control of live steam with intermediate water injection (see Figure 7) was chosen as a test case, since even this system although quite simple, is difficult to control.

The aim of the NMPC is to reach and to hold the set-point. Thus, the objective function to be implemented in the Optimica class of JModelica

$$\min_{m(t)} \int_{t_0}^{t_{end}} (T_{ref} - T_{mix}(t))^2 dt$$

contains the squared deviation between the tempera-

ture set-point T_{ref} and the current temperature $T_{mix}(t)$ of the steam cooled by water. The objective is subject to the model equations and a maximum constraint for the injected water mass flow $m(t)$. The dynamic model is composed of proven components already existing in the library SiemensPower. To meet the model requirements of JModelica, the functions of the water and steam properties have been approximated.

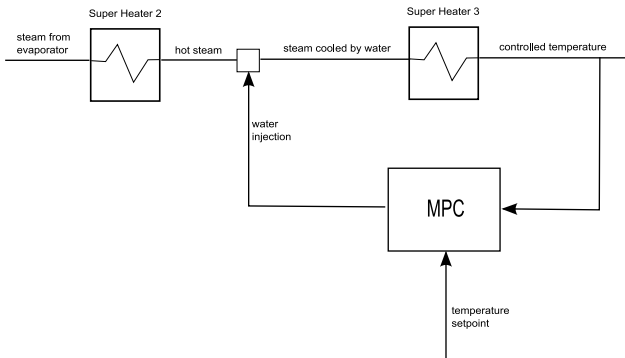


Figure 7: Temperature control of live steam with intermediate water injection.

For now, the NMPC problem has been implemented off-line. Instead of the real plant, the optimal control is applied to the plant model which simulates the behavior of the real process. This simulation provides the “measurements” of the states required for the feedback loop as described above.

Figure 8 shows the temperature profiles $T_{mix}(t)$ controlled by the NMPC (red) and a conventional controller typically implemented on present-day plants (blue). After some seconds the temperature controlled by NMPC reaches the set-point and is capable of maintaining this with negligible deviations. In contrast, the classical control is not able to avoid remarkable deviations from the set-point. Clearly, the NMPC is far superior to the conventional control and its on-line implementation should be seriously considered. However, the real time criterion has to be met. Running the application on a standard desktop it is not yet satisfied in all cases (see Figure 9).

In the example the control horizon is set to $T_c = 1s$, but the computation of the control profiles for one step needs more than one second, in particular for the former time horizons up to $n = 27$. As can be seen the computational effort for these early time horizons is higher than that of subsequent ones. This is presumably due to the fact that subsequent NMPC steps the temperature is already close to the set-point and the control input needs only slight modifications. Since for this quite simple model the computation time already represents a limitation, the computational effort will presumably be too large for more complicated or faster systems. Hence, in many cases

some performance improvements will be needed to run the applications on-line. An efficient algorithm is presented in [8] but it requires a more elaborate implementation which may be realized in the future.

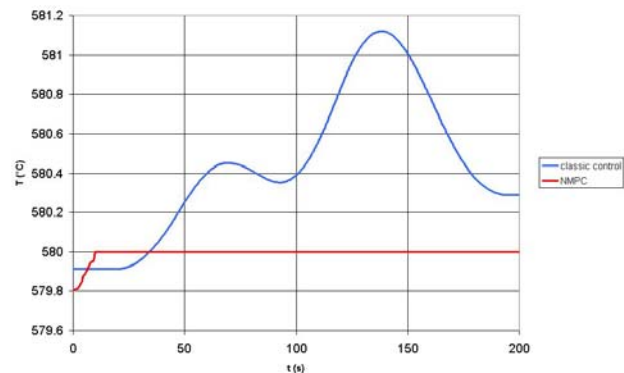


Figure 8: Temperature controlled by the NMPC and classic temperature control.

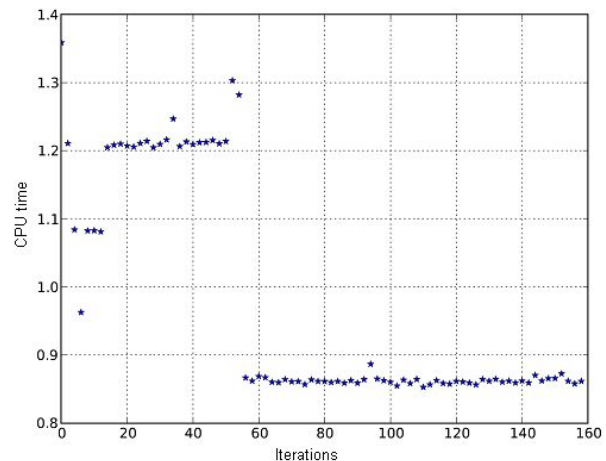


Figure 9: CPU time needed to solve the dynamic optimization problem for each moving prediction horizon with initial time $t_{init} = nT_c$.

5 Conclusions

In order to develop benchmark and realistic test cases, we implemented simple models for the systematic investigation of performance improvements of the OpenModelica tool chain. Some primary investigations have been carried out. Besides that the restricted functionality of OpenModelica with respect to water and steam properties has been supplemented by the usage of TTSE.

Parameter estimation and nonlinear model predictive control for fluid systems have been solved.

In fluid dynamics, we often have to deal with valves with discrete behaviour, flow reversal and phase transitions which considerably complicate the optimization of such problems. To overcome these difficulties we studied reformulation methods for hybrid

systems and showed their capability of tackling the problem of discontinuity of state trajectories and gradients which occur due to instantaneous transitions.

For nonlinear model predictive control a simple off-line example has been implemented. The model-based control proved to be superior to the recently applied classic control. Thus, the online application of the described solution should be seriously considered.

The support by German Ministry BMBF (BMBF Förderkennzeichen: 01IS09029C) within the ITEA project OPENPROD is gratefully acknowledged.

References

- [1] www.energy.siemens.com
- [2] www.openmodelica.org
- [3] www.jmodelica.org
- [4] Miyagawa, K., Hill, P.G., Tabular Taylor Series Expansion (TTSE) Method Based on IAPWS-IF97 Double Precision Enthalpy-Density Version, IAPWS Task Group TTSE, 2001-07-13
I. Weber, K. Knobloch, I. Kodl, H.-J. Kretzschmar, Test Report Of „Documentation and Software of TTSE Method applied to IAPWS-98 as an Example“, TTSE Evaluation Task Group, July 2002
- [5] 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.
- [6] Mynttinen, I. and Li, P.: A Reformulation Scheme for Parameter Estimation of Hybrid Systems. ESCAPE 2010, submitted
- [7] J. Till, S. Engell, S.Panek, O. Stursberg , “Applied hybrid system optimization: An empirical investigation of complexity,” *Control Eng. Pract.*, vol. 12, pp. 1291–1303, 2004.
- [8] B. T. Baumrucker, L. T. Biegler, “MPEC strategies for optimization of a class of hybrid dynamic systems,” *J. Process Contr.*, vol. 19, pp. 1248–1256, 2009.
- [9] J. M. M. Tamimi. Development of Efficient Algorithm for Model predictive Control of Fast Systems, PhD thesis, Technical University Ilmenau, 2011
- [10] JModelica 1.4.0 User Guide, <http://www.jmodelica.org/api-docs/usersguide/1.4.0>, Modelon AB 2010.
- [11] www.openprod.org

Modeling of hydraulic axial piston pumps including specific signs of wear and tear

Christian Bayer

Olaf Enge-Rosenblatt

Fraunhofer Institute for Integrated Circuits, Division Design Automation

Zeunerstrasse 38, 01069 Dresden, Germany

{Christian.Bayer; Olaf.Enge}@ eas.iis.fraunhofer.de

Abstract

Reliability of machines and facilities has played an important role since many years. Nowadays, attention is also paid to maintenance times. Maintenance standstills are to be reduced as far as possible. On the other hand, technical systems are subject to signs of wear and tear which are in general growing slowly and imperceptibly. The gradual abrasion of applied tools may lead to poor production tolerances or to a component's standstill. Hence, a condition-based maintenance strategy will be of increasing importance. Such a strategy requires a permanent condition monitoring during operation. To this end, reliable high-performance algorithms for signal processing, feature extraction, and classification are needed. Modeling the process of wear and tear may be useful to find the particular steps of the condition monitoring system's signal processing. This strategy was investigated by means of one very important device from automation engineering, a hydraulic axial piston pump. The procedure of getting signals by an appropriate Modelica model of the main parts of the pump is shown within the paper. Additionally, the manipulation process for the signals and the steps of classification are shortly presented to give an overview to the possibilities of model-based signal generation based on a Modelica model. The advantages of the multi-physics modeling language are emphasized because the axial piston pump model combines the mechanical and the hydraulic domain in a very efficient way.

Keywords: condition monitoring; classification; signal processing

1 Introduction

Long-lasting correct operation and the highest level of availability are important requirements concerning machines and facilities in today's industry. While reliability has played an important role since many

years, more and more attention has been paid to the times needed for maintenance processes in the last decade. The number of maintenance standstills has to be reduced to decrease the total cost of ownership (TCO). This is enforced by the increasing business rivalry of today's economy. On the other hand, technical systems are subject to a proceeding deterioration and often to a certain wear. But signs of wear and tear are in general growing slowly and imperceptibly. The gradual abrasion of applied tools may lead little by little to poor production tolerances or even to a standstill of a component or a complete production facility. Hence, a condition-based maintenance strategy will be of increasing importance. Such a strategy requires a permanent condition monitoring during operation based on efficient behavior analysis procedures. Condition monitoring systems analyze various measured signals using application-specific algorithms. In this context, reliable high-performance algorithms for signal processing, feature extraction, and classification are needed.

Modeling the process of wear and tear may be useful to find the particular steps of signal processing which are suitable for a certain condition monitoring system. This strategy was investigated by means of a hydraulic axial piston pump, which is a very important device from automation engineering and is still an intense subject of computer aided modeling [1, 2].

In section 2, two models of the standard pump are explained first. Both models combine a mechanical and a hydraulic sub-model. Physically motivated inter-modular connections are established. In section 3, a parameter optimization method is presented and discussed. This method is applied to fit unknown parameters of the mechanical sub-model. Measured time signals are used for comparison. Afterwards, some signs of wear are implemented in the model. This is presented in section 4. Two of the most problematic cases leading to the main failures are shown. Characteristic features deduced from Fourier transforms of vibration signals are used for classification of good pumps and pumps having failures.

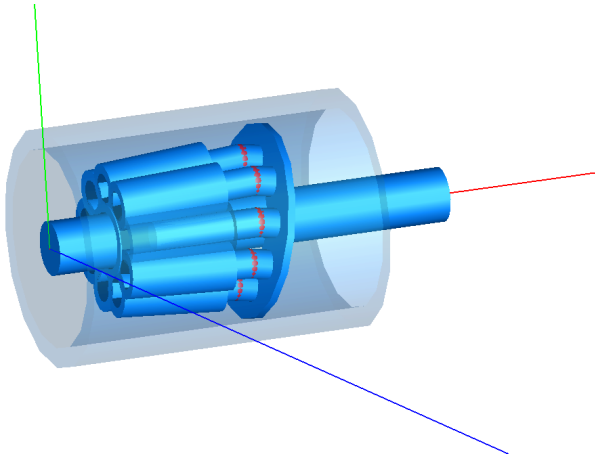


Figure 1: Schematic of the modeled axial piston pump with 9 cylinders. Only mechanical parts are shown.

2 Modelica model

2.1 The axial piston pump

For demonstration of the above mentioned condition monitoring approach we chose an axial piston pump. Those devices basically consist of a certain number of cylinders and pistons to pump hydraulic oil under high pressure conditions. The cylinders are evenly aligned around a rotating axis, whereas the pistons are attached to a tilted cam, which is called swash plate. Due to rotation, the pistons can perform an axial motion and thus pump the oil through inlet and outlet ports respectively. Figure 1 shows a rough schematic of the pump, which is captured from a 3D Dymola animation [3]. The oil flow is controlled by a distributor plate, which e.g. connects the outlet port to the cylinders with high oil pressure. The plate is not shown in Figure 1 as the flow control is implemented within the hydraulics model.

2.2 3D model

Modelica is capable of simulating multi-body dynamics as well as models based on user-specific differential equations. A good reference of the Modelica language can be found in [4].

Our initial model was a spatial model (3D) exploiting the MultiBody package for mechanics and using further differential equations for hydrodynamics. Unfortunately, a lot of parameters, for example certain friction coefficients or spring rates, are not known exactly, even not by the manufacturer. Hence, the model must be suited for parameter optimization algorithms in terms of computational effort. Since the 3D model is very complex, we reduced the model

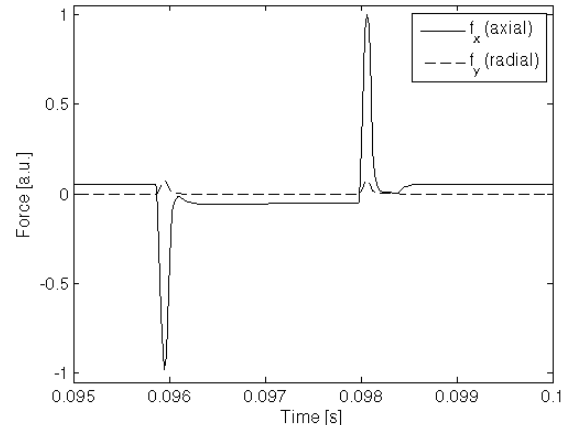


Figure 2: Comparison of axial and radial forces acting on the housing. The results are based on the 3D model.

to the very essential parts and developed a 1D equivalent. To justify this approach we evaluated the forces acting to the pump's housing. The device rotated at constant speed of 1500 revolutions per minute. Figure 2 shows axial and radial forces during a section of 1/9 of a full revolution of the pump. Obviously, the translational motion of the pistons is the main source of vibration, which is the value we want to measure and simulate. In an ideal case, the rotation itself does not contribute to vibration. Only the slight tilt of the cylinders against the rotational axis causes a radial force, which is directly related to the sine of the tilt angle. Since this angle is small, the radial force can be neglected. Therefore, the reduced model considers axial motion only and thus becomes a 1D representation.

Since we focused on the 1D model, the 3D implementation will not be explained in detail here.

2.3 1D model

The reduced mechanical model is depicted in Figure 3. Each element of the schematic relates to a dedicated Modelica sub-model, which in turn is described by certain differential equations and an individual set of parameters. Our 1D model complies with the conventions of Modelica's translational mechanics library and we also used a few basic components thereof. The elements are chained exactly as shown in Figure 3 and form lines or parallel lines respectively.

The housing of the pump is modeled as a simple mass, which embraces all other parts of the model. It is therefore connected in parallel to both sides of the pump's interior element chain. The mass is not fixed and its movement gives rise to the acceleration value and thus vibration. On the left side of the chain a

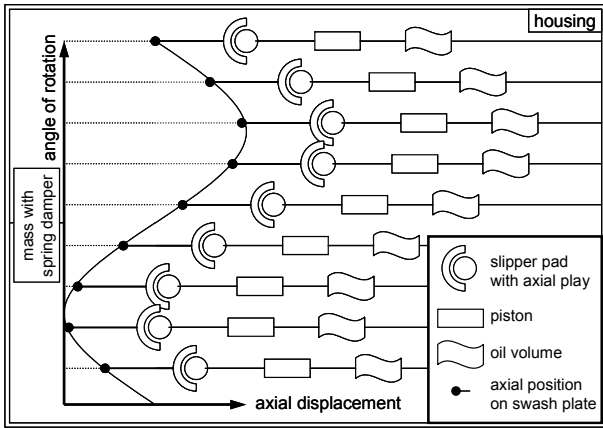


Figure 3: Reduced 1D model of the axial piston pump.

mass with spring damper follows the housing. It sums up certain masses inside the pump and connects them to the housing by means of a spring. This indeed approximates the real assembly of the pump.

The next element in the row is the tilted swash plate, represented by the sine curve in Figure 3. It causes all pistons to follow a sinusoidal motion with individual phase offset. This effect is implemented by time dependent displacements as follows:

$$\begin{aligned}
 0 &= F_a + F_b \\
 s_{rel} &= s_b - s_a \\
 &= \hat{s}(1 - \cos(\varphi)) \\
 \varphi &= 2\pi \frac{N}{60} t + \varphi_{piston}
 \end{aligned} \tag{1}$$

F denotes forces at the sub-models flanges and s is the distance between flanges or the position of them. φ is the absolute revolution angle of the pump. The swash plate model can only drive one of the 9 pistons with specific phase offset. At this point the model chain breaks up into 9 individual lines in parallel. Each line consists of a swash plate model followed by slipper pad, piston and the cylinder oil volume. All 9 lines finally merge into the right hand frame of the housing mass.

The connection between piston and swash plate is established by slipper pads. They offer a tight connection to the piston with small play, which extends as a result of wear. The sub-model allows a small distance of free motion until the piston hits one of the two limiting boundaries. Once in contact with the limitation, the model switches to a damped spring mass system behavior with spring rate D , as can be seen in the equations below.

$$\begin{aligned}
 F &= -F_a = F_b \\
 s_{rel} &= s_b - s_a \\
 F &= \begin{cases} s_{rel} < 0 & : R_1 \frac{\partial}{\partial t} s_{rel} + D_1 s_{rel} \\ s_{rel} > s_{gap} & : R_2 \frac{\partial}{\partial t} s_{rel} + D_2 (s_{rel} - s_{gap}) \\ otherwise & : 0 \end{cases} \\
 R_1 &= \begin{cases} v > 0 & : 0 \\ otherwise & : R_1^* \end{cases} \\
 R_2 &= \begin{cases} v < 0 & : 0 \\ otherwise & : R_2^* \end{cases}
 \end{aligned} \tag{2}$$

R corresponds to friction coefficients and v to the velocity of the related piston. It turned out, that the simulation results improve, if there is no friction when the piston pulls apart from a boundary. The equations for R_i therefore depend on the sign of v .

The piston itself is a simple mass of length L with friction and inertia. Stribeck friction is not used in the model as it would introduce too many unknown parameters. The piston equation reads

$$\begin{aligned}
 F_a + F_b &= m \frac{\partial^2}{\partial t^2} s_a + R \frac{\partial}{\partial t} s_a \\
 s_{rel} &= s_b - s_a = L
 \end{aligned} \tag{3}$$

The sub-model of oil volume V approximates the conditions within a cylinder and finally links hydraulic quantities to the mechanical part of the model. In [5] this was done by incorporating two different software packages for hydraulics and mechanics. We used Modelica for both physical domains to merge them into one model.

Pressure p inside a cylinder causes fluid flow q through the ports as well as force to a piston's flange. The oil is considered as compressible fluid with bulk modulus K . It flows either through the inlet or the outlet port, which depends on the relative angular position of cylinder block and distributor plate. Both ports are modeled as valves with variable cross sections and, thus, have time dependent hydraulic conductance G . For a single cylinder the valves' cross sections are shown in Figure 4. Other cylinders experience the same characteristic, but with a phase offset. The model assumes turbulent flow through a valve [6], which finally leads to the equations

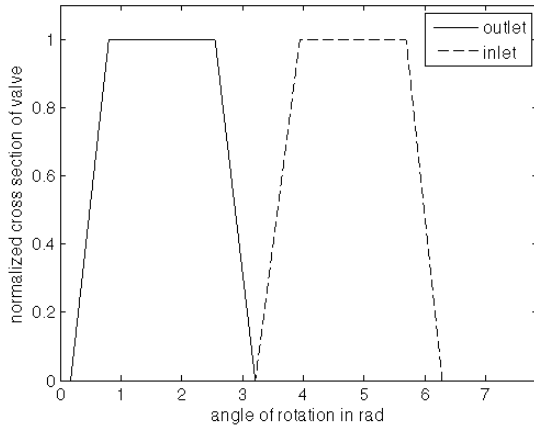


Figure 4: Representative cross sections of inlet and outlet valves for a single cylinder. The graph shows conditions for one revolution of the pump.

$$\begin{aligned}
 s_{rel} &= s_b - s_a \\
 V &= s_{rel} A_{piston} \\
 q &= G_{out} \operatorname{sgn}(P_{max} - p) \sqrt{|P_{max} - p|} + \dots \\
 &\quad + G_{in} \operatorname{sgn}(P_{min} - p) \sqrt{|P_{min} - p|} \\
 Q &= q + A_{piston} \frac{\partial}{\partial t} s_{rel} \\
 \frac{\partial}{\partial t} p &= K \frac{Q}{V} \\
 F &= F_a = -F_b = p A_{piston}
 \end{aligned} \tag{4}$$

The time dependent cross sections of inlet and outlet are provided by the manufacturer. They were implemented by using Modelica's look-up table component. As can be seen from equation (4), the oil volume sub-model has two frames with variables s and F . Hence, it can be easily connected to our chain of other mechanical translational components.

3 Parameter optimization

3.1 Method

Each sub-model has an individual set of parameters. Some of them are known, as is typically true for geometric parameters. Others have to be estimated or can be found by optimization. For this purpose, several measurements with at least one undamaged pump have to be done. With this reference data, an optimization algorithm was used to find an appropriate parameter set. This is a common method and was also applied in [7] for example. We used MATLAB® [8] for an automated parameter sweep in Dymola. The goal function for minimization is sim-

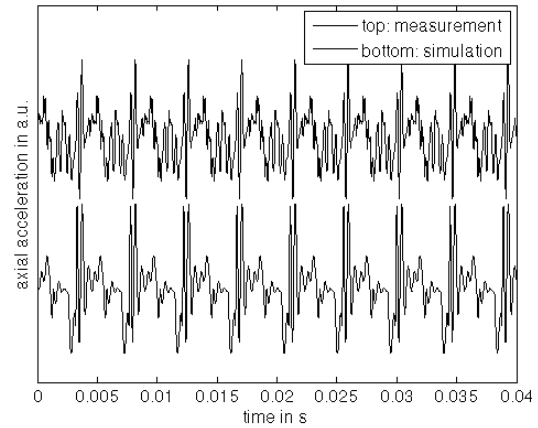


Figure 5: Comparison of simulation results with optimized parameter set and averaged measurement. The signals correspond to the time resolved acceleration of the housing in axial direction. The graph shows one revolution of the pump.

ply the deviation of the simulation result from the averaged measurement signal. The measurand equals in both cases the acceleration of the housing in axial direction. Unfortunately, the number of parameters is quite high, which impedes a reasonable optimization process. Within our reduced 1D model we identified seven parameters, which significantly influence the result and were used for optimization: The simulation is very sensitive to the interaction of piston head and slipper pad. Already four parameters can be extracted from this knowledge, two for each limiting boundary. When hitting a boundary, the piston will penetrate into the material of the slipper pad, squeeze the oil film between pad and swash plate or strain the pad. An easy but efficient way is to assume a spring-like model with damping for that process. This gives two parameters for each boundary, namely spring rate and friction coefficient. Another important parameter is the friction of the piston within the cylinder. The last two parameters belong to the spring-mass system, which is located between housing and swash plate (see Figure 3).

One problem of the optimization is that there is no unique ideal parameter set. The results differ with the initial parameters. To account for this, we also implemented a random sweep of starting values. Once a good parameter set is found, the actual optimization is performed. The algorithm used here was the simplex search method.

3.2 Results

In spite of the complicated optimization process, we found a set which fits the real measurement quite well. Figure 5 shows the result. The signal itself has

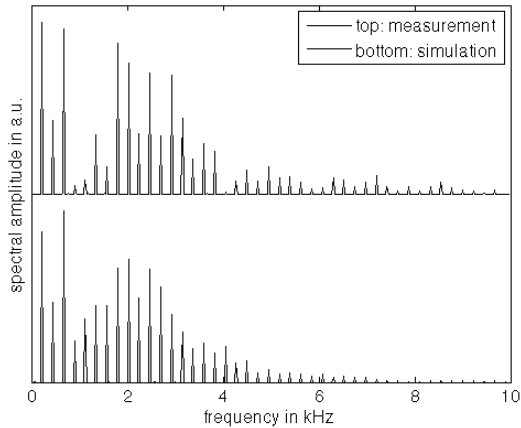


Figure 6: Spectra of simulation result with optimized parameter set and averaged measurement. The spectra correspond to the Fourier transform of the acceleration of the housing in axial direction.

some clear characteristics. During one revolution of the pump, each piston accomplishes one period of its sinusoidal movement. This leads to 9 equally shaped pulses in the time signal. Both, simulated and measured signal, show this characteristic and are comparable.

The axial piston pump is a rotating device. For condition monitoring of this special application the spectrum and evaluation of harmonics is far more interesting than the time signal. Hence, the consistency of simulated and measured signal in the frequency domain is very important. Figure 6 compares the results. Due to the 9 pistons and the rotational frequency of 25 Hz, harmonics with a spacing of 225 Hz can be observed. Both spectra show a similar distribution. However, within the measurement data harmonics 4 to 7 seem to be rather suppressed, which holds for all measurements, even for different pumps with varying signs of wear. This effect might be caused by the experimental setup. The pump is attached to a motor and other machinery which can vibrate as well. Our model does not consider the exact environmental conditions and is not able to reproduce the harmonics suppression. But it turned out, that we can simply ignore these harmonics for classification.

4 Modeling defects

4.1 Introducing defects

The main purpose of the simulation is the estimation of measurement signals for worn pumps. Once the model parameters are optimized for an undamaged device, they can be modified slightly to introduce

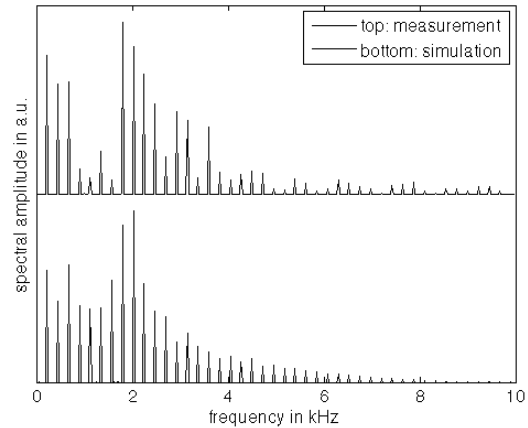


Figure 7: Spectra of measurement signals of a worn pump with axial piston play and simulation result.

defects. As a matter of course this requires the parameters to be related to certain signs of wear.

We investigated two types of defects. One of them is cavitation pitting, which influences the hydraulic conductance of the cylinder's valves. Basically, the cross section of each valve changes slightly with gradual abrasion. Worn pumps give suitable information on the modified cross section and we simply exchanged the characteristic area functions, shown in Figure 4. However, our results suggest that a reasonable change of the functions has only a minor effect to the axial vibration within the considered frequency range. This is consistent with measured data, where the difference is present but comparably small. In the following we focus on a more significant defect.

Each piston is attached to a slipper pad with a certain amount of play. This small gap increases with time and leads to a defect referred to as axial piston play (app). We introduced this effect in the slipper pad sub-model by increasing the play parameter and decreasing one of the spring rates, which models the strain of the pad when pulled by the piston. Figure 7 compares measurements of a worn pump with axial piston play and our simulation results. Since the classification is done in the spectral domain, only the Fourier transforms are shown. Ignoring harmonics 4 to 7, we can see a very similar trend of change in both the simulation and measurement. The first 3 harmonics drop in amplitude, while others near 2 kHz increase. Based on such effects unique features for classification can be found.

4.2 Classification

The classification method used here is based on the first 17 harmonics of the signal's spectrum excluding harmonics 4 to 7. Several statistical parameters, like

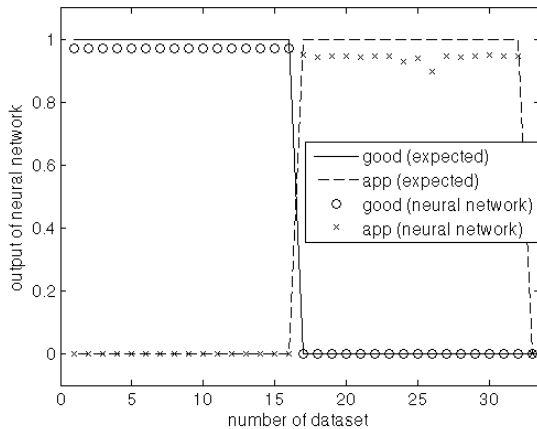


Figure 8: Classification result for an undamaged pump (good) and axial piston play (app). Lines denote the expected output of the NN for ideal classification. Circles and crosses denote classification results of measured data.

standard deviation or skewness were defined and applied to specific harmonics arranged in groups of at least 7. If the groups are thoroughly composed, meaningful features can be extracted.

With subject to complexity a neural network (NN) was employed to classify the condition of pumps. A great advantage is the ability to train such a network with simulation data instead of experimental data, which would cause higher effort. However, in most cases measurement data of undamaged devices should be available. It is therefore reasonable to train the NN with those measurements and in addition with model data of all simulated defects and conditions. In our case we modeled an undamaged pump as well as one with axial piston play and presented both simulation and measurement (undamaged) data to the NN for training. The NN accepted 12 features as input and had 3 output neurons. Thus, 3 classes could be separated, but we used only two of them for an undamaged pump and axial piston play.

The classification results are presented in Figure 8. Only measurement data was presented to the NN for testing. A value of 1 at an output neuron means that a signal is strongly related to the specific class. Smaller values indicate a lower probability of class membership. The classification works well for our presented model and is suitable for a CMS.

5 Conclusions

In this paper we presented a model-based method to support the development of condition monitoring systems (CMS). The basic idea is to replace an experimental setup by simulation or at least to reduce

the experimental effort during development of a CMS. For demonstration an axial piston pump with at least two signs of wear was modeled. The simulation results agree comparatively well with experimental data. Modeling has several advantages compared to experimental setups. The investigation of defects is much more flexible and one is able to find correlations between those defects and detected signals. This approach also facilitates the development of improved signal processing techniques and the generation of better features for classification.

Acknowledgement

This project was funded in part by the German Federal Ministry of Education and Research (BMBF). Associated partners are GEMAC GmbH (Chemnitz), Fraunhofer Institute IIS (EAS, Dresden) and Lenord+Bauer & Co. GmbH (Oberhausen).

References

- [1] Ming Liu, *Dynamisches Verhalten hydrostatischer Axialkolbengetriebe*. Bochum, Germany: PhD thesis, Institute Product and Service Engineering, Ruhr-Universität Bochum, 2001
- [2] Liang Chen, *Model-based fault diagnosis and fault-tolerant control for a nonlinear electrohydraulic system*, PhD thesis, TU Kaiserslautern, 2010.
- [3] Dymola 7.3, Dassault Systèmes
- [4] Fritzson P., *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*, IEEE Press, 2004
- [5] A. Roccatello, S. Mancò and N. Nervegna, *Modelling a Variable Displacement Axial Piston Pump in a Multibody Simulation Environment*, *J. Dyn. Sys., Meas., Control*, 129(4):456, 2007.
- [6] Zoehl H., Kruschik J., *Strömung durch Rohre und Ventile*, Springer-Verlag Wien, Austria, 1978
- [7] R. Petrovic, *Mathematical Modeling and Experimental Research of Characteristic Parameters Hydrodynamic Processes of a Piston Axial Pump*, *Journal of Mechanical Engineering*, 55(4):224-229, 2009
- [8] MATLAB® 2010a, The MathWorks, Inc.

Detailed Model of a Hydromechanical Double Clutch Actuator with a Suitable Control Algorithm

Sebastian Nowoisky Chi Shen Clemens Gühmann
 Technische Universität Berlin

Chair of Electronic Measurement and Diagnostic Technology

Sekr. EN 13, Einsteinufer 17, 10587 Berlin

{sebastian.nowoisky@, c.shen@mailbox., clemens.guehmann@}tu-berlin.de

Abstract

This paper presents the detailed model of a double clutch actuator with a suitable control algorithm. Firstly, there is an introduction into the theory of a double clutch transmission and the aim of this project. The simulation model with a Dymola[®] and a MATLAB[®]/Simulink[®] part is discussed. The library of a vehicle model with a highly detailed hydromechanical clutch is introduced, which includes models with different levels of detail. The modeling of the hydraulic and the mechanic parts of the clutch actuator is discussed, concentrating on the problem of determining the parameters of the actuator modules e.g., the hydraulic valves. Some parts could not be used from existing Dymola[®] libraries, in those cases, new models are created based on Modelica code.

A translational lever is pictured with its source code. Furthermore the non-linear behavior of the clutch actuator and control design is described. To verify this model and the suitable closed loop controller, the algorithm is tested with an up-shift cycle in a vehicle model with a double clutch transmission. The simulation results are presented with a global view of the driver inputs, the speed, the torque of the vehicle model and the gear status. Additionally the local view of the clutch actuator is shown with the cylinder pressure, the clutch position and the clutch capacity (torque). Finally there is a summary and an outlook on the further development of this library.

keywords: double clutch transmission, powertrain, clutch, hydraulic, transmission actuator

1 Introduction

A modern powertrain with a double clutch transmission (DCT) combines the comfort of an automated transmission with the efficiency of a manual transmission. Moreover, an uninterrupted shift process, a very good ride comfort and easy handling are considerable characteristics of an automated transmission with double clutches [1, 2].

The double clutch transmission is similar to the hardware design of a manual transmission. Instead of one input shaft for the clutch, a double clutch transmission has two separated shafts with odd and even gears. The two clutches can be continuously changed from one input shaft to the other. The result is an automated and uninterrupted shift process. The transmission conforms to different driving situations. In other words, the shift process changes, depending on the driving situation.

The clutch control system is very important in relation to driving behavior. When the vehicle accelerates very fast, the clutch is used to control the speed by means of a higher clutch capacity¹. For effortless driving, the input torque is followed by the torque capacity of the clutch. In this case, the speed is controlled by the engine controller.

Nowadays software functions for drive train applications are developed and tested in the model based V-process [3]. To develop software for the nonlinear and complex hardware a detailed model is necessary.

The clutch actuator consists of several non-linear subsystems e.g. the hydraulic valves or the mechanical disc spring of the clutch. These parts have a hysteresis behavior. To get a detailed clutch model, the existing libraries e.g. Powertrain- or Vehicle Interfaces-Library have been expanded. Important variables like position

¹The clutch transfers the torque only, and this torque is termed capacity.

and torque (capacity) have been integrated.

In this paper we present the modeling, the control and the simulation of a double clutch transmission with a highly detailed model of hydromechanical clutch actuator. First, the model of the vehicle library with the gear set is presented which comprises the hydraulic and mechanic parts. In section 3, the control algorithm for the hydromechanical clutch actuator is discussed. Then, in section 4, the suitability of the models and of the controls is demonstrated in a simulation runs.

2 Modeling

The modeling of the double clutch transmission and the clutch actuator is based on a real transmission for mid-size cars. This transmission is a seven speed transmission with a dry double clutch and a torque load up to 250 Nm.

For the modeling, two common tools are used. The code of the Transmission Control Unit (TCU) and the Engine Control Unit (ECU) had been developed with MATLAB[®]/Simulink[®]. The powertrain and the drive environment is modeled with Dymola[®]. For the interaction between both models, an interface is defined. The interaction between the tools is pictured in figure 1. For the simulation the Dymola[®] powertrain model is included as an embedded s-function in the MATLAB[®]/Simulink[®] model. The model library

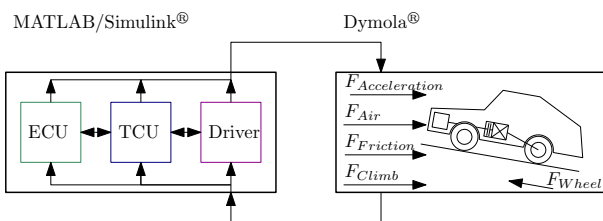


Figure 1: Structure of the powertrain and control model

has common parts such as the User Guide with information for the users and the Examples or the Interfaces. The Interfaces are models with an input and output description, for each sub-module of the transmission model. In the Engine module, there is an engine model based on a simple look-up table. The Transmission module has several sub modules, which are shown in the second row of figure 2. The Library is used for research and teaching, hence there are two different designs of the transmissions in the Gearset module. Some models have different levels

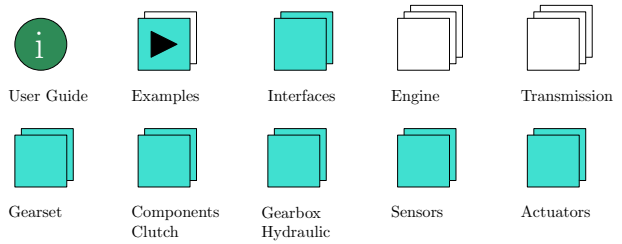


Figure 2: Modules of the DCT Vehicle Library

of detail. For example, a pair of gears exists as a plain and as a complex model. The plain model consists of the ratio and the inertia of the toothed wheel. The complex model gear losses and bearing friction are considered. The friction element corresponds to the current transmission design. The Components Clutch module is a top level module for this topic. This module implies several clutch models with a simple ideal model and a detailed hydromechanical model. The Gearbox Hydraulic module includes the models with the hydraulic components of the transmission, for instance the hydraulic supply, valves and the clutch cylinder. Furthermore the clutch models include mechanical parts such as the lever or the disc spring. These mechanical parts are components of the Actuator module. In the Sensor module there are several types of sensors to measure the torque, the speed and the acceleration.

Some blocks of the vehicle model are derived from the Powertrain library, e.g., the vehicle model or the drive environment. The Interfaces are derived from a former research project [4]. The hydraulic part is modeled with the Hydraulic Library (HyLib) from Modelon [5].

2.1 Hydraulic Components

The hydraulic components control and moves the clutch plates in the transmission. Because of, the high energy density, a hydraulic supply is used for mid-size cars [6]. An electrical external toothed gear pump supplies the actuator with the hydraulic energy. The pump is switched on and off depending on the current pressure level with a hysteresis switch. The typical pressure supply for a small transmission is approximately 50 bar. This pressure is limited by two 'three port valves' to a level of approximately 16 bar. This pressure level oscillates in a smaller range than the original bang bang controlled hydraulic supply. Furthermore these valves separate the pressure for the odd

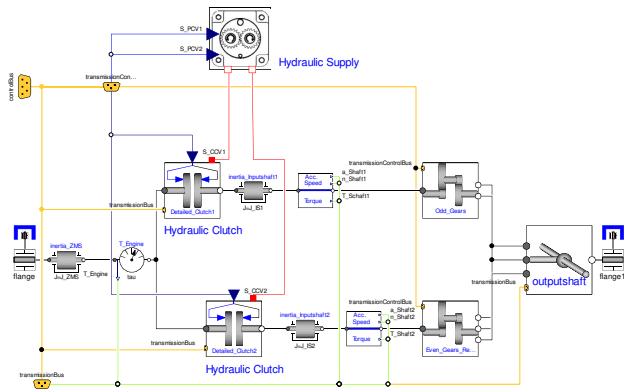


Figure 3: Model of the transmission with replaceable modules

and even gears and the two clutches. Behind this, six other valves control the actuators to set the gears or the clutches. The position between the friction plates of the clutch is set by a single acting cylinder. The model of the double clutch transmission with the hydraulic supply is shown at the top of figure 3. Underneath the hydraulic supply, the detailed model of the hydraulic clutch system can be seen. Each system is replaceable by using the defined interfaces. The parameters correspond to values from literature of similar transmissions [7, 8, 9]. Some parameters were determined at the transmission by means of measurement.

2.2 Mechanic Components

The input of the mechanic interface for the hydraulic clutch actuator is a lever that changes the position of the clutch. The rod of the hydraulic cylinder moves outward when the chamber is filled with oil and simultaneously the rod presses against a lever. The force is transferred from the rod to the clutch disc. This lever system described above, is used to control the double clutch capacity. Without hydraulic pressure the clutch is automatically opened. In this instance, the clutch capacity can not be built up anymore, in order to protect the clutch system.

The model of the clutch lever is made up of two rods. One rod is fixed on the top, and the other is fixed in the middle. Figure 4 shows two separately modeled levers.

According to the different lengths of the rods, the model of the rod with top-fix and middle-fix are expressed by Modelica as follows:

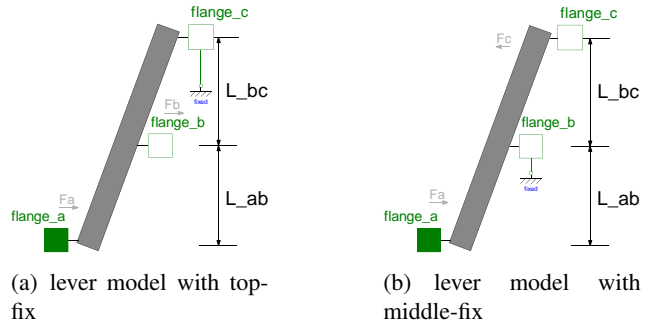


Figure 4: The two sub models of the lever

Code example for Lever_TopFix (figure 4a)

```

model Lever_TopFix
  Modelica.Mechanics.Translational.
    Interfaces.Flange_a flange_a
  Modelica.Mechanics.Translational.
    Interfaces.Flange_b flange_b
  Modelica.Mechanics.Translational.
    Interfaces.Flange_c flange_c
  Modelica.Mechanics.Translational.
    Components.Fixed fixed
  parameter Modelica.SIunits.Length L_ab;
  parameter Modelica.SIunits.Length L_bc;
  equation
    flange_b.f = -flange_a.f * ((L_ab+L_bc)/L_bc);
    flange_b.s = flange_a.s * (L_bc/(L_ab+L_bc));
  end Lever_TopFix;
  
```

The code for the lever model with the middle fix is similar to the top-fix model, only the equation is different. Every flange of the model provides two types of information: one is the displacement which is described in model with the character - 'flange_x.s', the other is the force, which is described with the character - 'flange_x.f'.

The force within the clutch is transferred by the two levers having two pivot points. The second clutch is modeled similar to the first clutch. In contrast to the first model, the two rods are modeled with the top-fix model. Figure 5 shows an example for modeling the lever mechanism of the first clutch. This lever model consists of two sub-models, which are described in figure 4. The first model (top-fix) output port is connected with the second model (middle-fix) input port. The lever works against the disc spring which has a non-linear behavior [10]. The behavior of the disc spring is modeled by some look-up tables. Within the front range of the clutch position the lever presses the pressure plate against the central plate. The clutch disc is between both plates. As long as the clutch disc does not touch the central plate the spring force increases linearly. When the clutch disc touches the central plate (touch point) the spring force starts to decrease. In or-

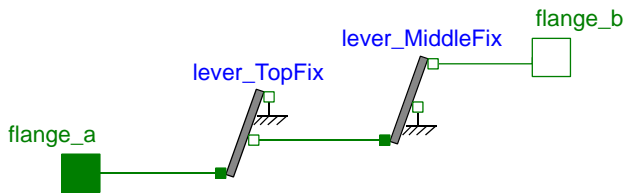


Figure 5: Lever mechanism for clutch 1

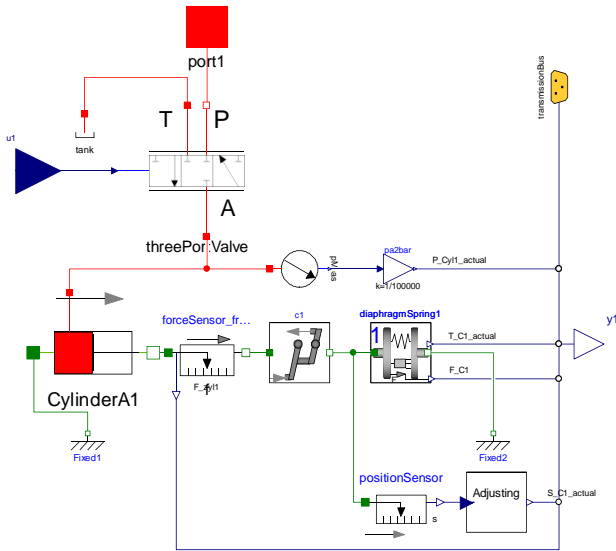


Figure 6: Detailed Clutch Model

der to disengaging the clutch, the parameters change to a lesser value because of the hysteresis of the disc spring [11].

Both the hydraulic one and the mechanic clutch system are presented in figure 6. The output of the model is the position of the lever. Furthermore the pressure of the cylinder, the force and the torque of the clutch system are also shown.

3 Control

There is an air clearance between the clutch disc and the clutch housing. Before the clutch torque builds up, the hydraulic system has to be filled with oil to generate the necessary pressure in the clutch cylinder. Then the clutch plates come together up to the touch point and the clutch torque can be built up. The time lag of the clutch system is minimized by using the pre-fill function. This function is active from a shift request up to the touch point of the clutch (as shown in figure 7 phase II).

The behavior of the actuators is derived from the literature [12]. The control of the clutch can be divided

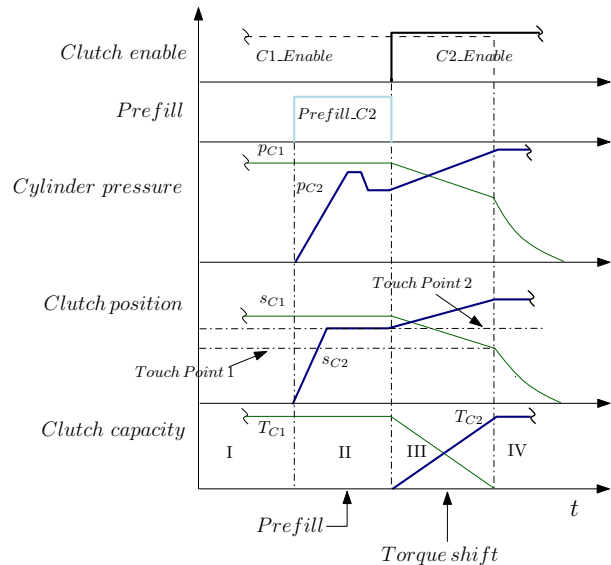


Figure 7: The basic function of the controller during build up and reduction of the clutch torque

into four stages (as shown in figure 7):

1. Clutch deactivation or relief condition (Phase I - C1);
2. Pre-filling (Phase II - C2);
3. Clutch activation (Phase I, II, III - C1, or Phase III, IV - C2)
4. Emptying of the clutch cylinder (Phase IV - C1).

Figure 8 shows the control structure of the clutch. The output signals from the clutch model are cylinder pressure (p_{Cyl_actual} [bar]) and clutch position (s_{C_actual} [m]). The command signal is the input for the clutch model. This signal is a normalized valve position signal ($S_{CCVx} [-1 \dots 1]$), which is combined with a non-linear spool dynamics block. The spool dynamics block contains hysteresis, friction and a limit of the spool velocity [5]. The volume flow of the hydraulic oil is controlled by the spool position.

The inner loop is used to control the cylinder pressure and the outer loop controls the clutch position (as shown in figure 8). Both loops are realized by a PI controller. The control parameters for the inner loop circle have been empirically adjusted. The $T\Sigma$ -method provides the opportunity for calculating the parameters of the outer loop circle [13]. To shorten the time for the parameter settings, the nonlinear system was linearized. With the linearized system the control parameter could be determined and it was possible to check the stability.

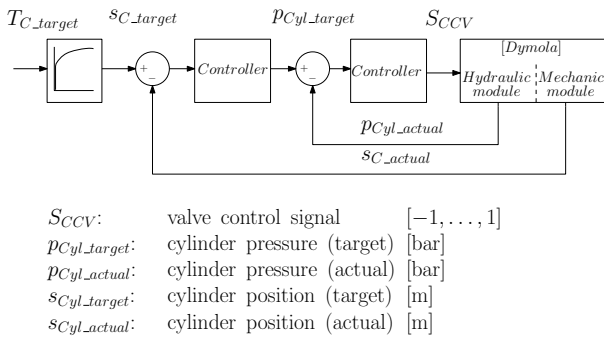


Figure 8: The control design of the clutch model

The control strategy is based on a condition-based cascade control. This means that the pre-filling and the emptying of the clutch cylinder takes place only when the inner control system is active. If the clutch status is enabled, the overall cascade control system is in processing mode. To switch continuously from the "pre-fill" status to the "clutch enable" status, the start value of the integrator of the outer controller is set to the last pressure value of the pre-fill phase. An anti-windup process is used for counteracting the windup in the inner controller, after pressure emptying of the clutch cylinder. The valve control signal returns as quickly as possible to the operation state (S_{CCVx} control signal is closed to zero) as shown in figure 10. Additionally, a dither generator for the valve has been developed for overcoming the stick-slip-effect, which could occur due to the inaccuracy of the valve control.

4 Simulation Results

In section 2 the simulations model with the interface and the implemented control algorithm were presented. Now the clutch model, the suitable control algorithm, the performance and the functional capability of a basic shift strategy are shown in a simulation run. The simulation can be executed with a driver model, so it is possible to drive the driving cycles like the New European Driving Cycle (NEDC) [14]. The model inputs are a boolean ignition signal and two real values for the acceleration- and brake-pedal. Figure 9 shows the first simulation result containing the input signals accelerator-, throttle-position and the brake-pedal position. The throttle position is controlled by the engine control unit. The driver starts the engine with the ignition key. After the brake pedal is released by the driver and the acceleration pedal is pushed. This happens as long as the TCU has shifted up to the seventh gear. The second row of figure 9 shows the

engine speed (n_{Engine} [rpm]) and the speed of the two input shafts (n_{Shaft1}, n_{Shaft2} [rpm]). The engine speed has to follow the torque transferring input shaft. If a higher gear is required, the synchronization releases the actual gear and engages the required one. This happens before the torque transfer begins. The next sub figure shows the induced engine torque with engagement (T_{Engine_i} [Nm]), the engine torque (T_{Engine} [Nm]) and the torque of the input shafts ($T_{Schaft1}, T_{Schaft2}$ [Nm]). The next row of the figure illustrates the target clutch capacities ($T_{C1_target}, T_{C2_target}$ [Nm]) and the actual clutch capacities ($T_{C1_actual}, T_{C2_actual}$ [Nm]). The last sub figure shows the signal *desired gear* and the *current gear*.

The vehicle accelerates and shifts up to the seventh gear. Each of these shift processes is accompanied by a continuous switch of the clutch capacities. The engine speed is reduced by the throttle position control, which is a submodule of the ECU. If sporty behavior is requested by the driver, the clutch control could be used to execute the speed-regulation. The engine intervention is controlled by the transmission control unit with a defined torque interface between these two control units. Figure 10 shows the detailed results of the third and the fourth gear for an up-shift process. On the left side the simulation results for the first clutch are shown. The right figure shows the results for the second clutch. In the first row, there are the normalized input signals of the hydraulic three port valves ($S_{CCVx} [-1 \dots 1]$), followed by the results for the cylinder pressure signals (p_{Cylx_actual} [bar]). With the changing of the pressure signal there is a change in the position of the clutch (s_{Cx} [m]), which is shown in the next row of the results. In the last row, the target and the actual clutch capacity are shown ($T_{C1_target}, T_{C2_target}$ [Nm]).

The results of the simulations show the expected behavior for the hydraulic clutch system. The results especially show the relationship between the clutch position, cylinder pressure and clutch capacity. The quality of the model could not be determined without a comparison with measurements data from the transmission. The duration for the pre-fill function and the shift process is related to other DCT application [15].

5 Summary and Outlook

A detailed model of a hydromechanical double clutch actuator with a suitable control algorithm was presented. The model was integrated into a double clutch

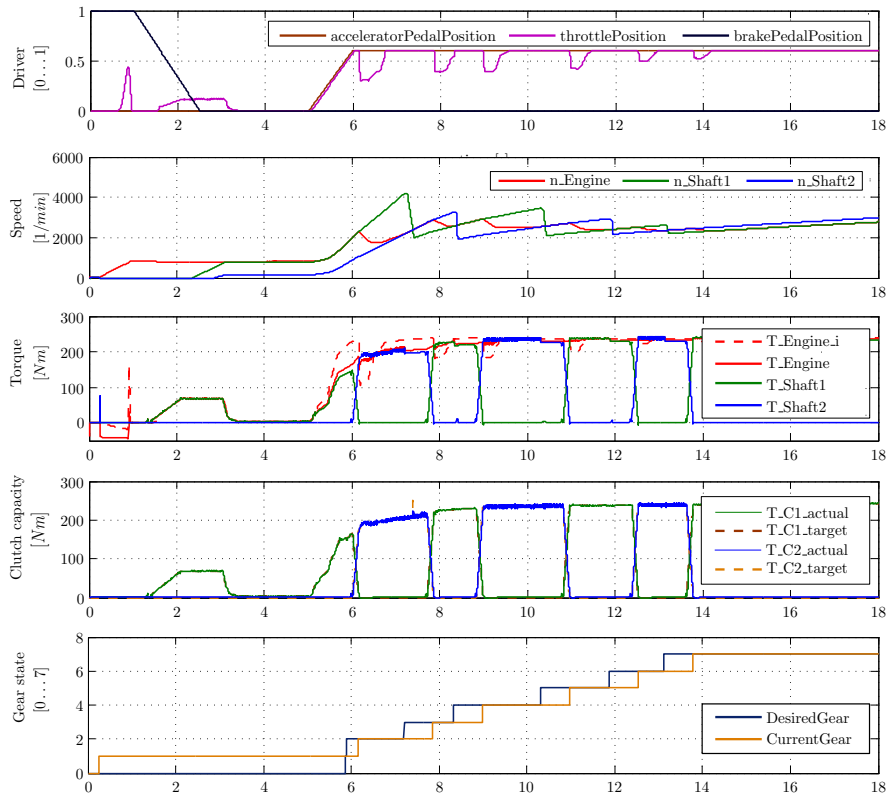


Figure 9: Simulation result of a up-shift cycle

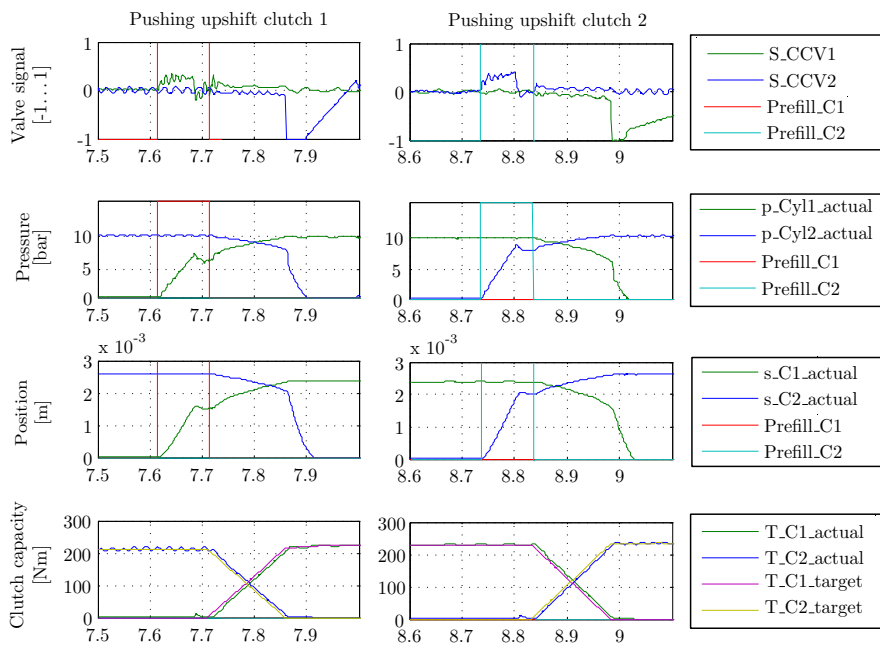


Figure 10: Detailed results of a shift process for the hydro-mechanic clutch actuator

transmission model with seven speeds. The main focus was the modeling of the mechanical lever system. The detailed clutch model provides engineers the opportunity to develop new control algorithms in model-in-the-loop simulations. As example a cascaded PI controller was designed. The clutch model, the suitable control algorithm, the performance and the functional capabilities of a basic shift strategy were shown in a simulation run. The verification of the simulation models with the real transmission is in progress. Therefore a new transmission test bench at the TU Berlin, Chair of Electronic Measurement and Diagnostic Technology is used. At the moment the extensible hydraulic interface is used to implement a detailed synchronization model for the gear shift process. With this synchronization model in combination with the clutch model the calibration of the shifting and clutch algorithms in a model-in-the-loop simulation is feasible.

References

- [1] U. Eggert, C. Kraus, M. Leibbrandt and K. Bernemann. The new family of Powershift Transmissions at GETRAG FORD Transmissions GmbH. In *VDI Berichte 1943, Getriebe in Fahrzeugen 2006*, page 289...308, Düsseldorf, 2006. VDI Verlag.
- [2] S. Rinderknecht and U. Knödel. Evolution and Future Potential of Passenger Car Transmissions in Layshaft Design. In *VDI Berichte 2029, Getriebe in Fahrzeugen 2008*, Düsseldorf, 2008. VDI Verlag.
- [3] C. Gühmann. Model-Based Testing of Automotive Electronic Control Units. In *3rd International Conference on Materials Testing: Test 2005*, 2005. Nürnberg.
- [4] H. Isernhagen and C. Gühmann. Modelling of a Double Clutch Transmission with an Appropriate Controller for the Simulation of Shifting Processes. In Bernhard Bachmann, editor, *Proceedings of the 6th International Modelica Conference*, pages 333–339, Bielefeld, Germany, 2008. Modelica Association and University of Applied Sciences Bielefeld.
- [5] Hylib (2009), version 2.7. Product help, Modelon, 2009.
- [6] G. Lechner, H. Naunheimer and S. Day. *Automotive Transmissions: Fundamentals, Selection, Design and Application*. Springer, Berlin, Heidelberg, New York, 1st edition, 1999.
- [7] F. Rudolph, M. Schäfer, A. Damm, F.-T. Metzner and I. Steinberg. Das innovative 7-Gang-Doppelkupplungsgetriebe für die Kompaktklasse von Volkswagen. 28. *Internationales Wiener Motoren-symposium*, 639:242–264, 2007.
- [8] D. Findeisen. *Ölhydraulik: Handbuch für die hydrostatische Leistungsübertragung in der Fluidtechnik*. Springer, 2006.
- [9] W. Staudt. *Kraftfahrzeugmechatronik*. Bildungsverl. EINS, 2007.
- [10] U. Kiencke and L. Nielsen. *Automotive Control Systems, For Engine, Driveline, and Vehicle*. Springer, Berlin, Heidelberg, New York, 2nd edition edition, 2005.
- [11] E. Kirchner. *Leistungsübertragung in Fahrzeuggetrieben*. Springer Verlag, Berlin, Heidelberg, New York, 2007.
- [12] K. Reif. *Automobilelektronik: Eine Einführung für Ingenieure*. ATZ-MTZ-Fachbuch. Vieweg+Teubner, Wiesbaden, 3rd edition, 2009.
- [13] L. Holger and W. Wendt. *Taschenbuch der Regelungstechnik*. Verlag Harri Deutsch, 2009.
- [14] Emission Test Cycles. Summary of worldwide driving cycles. www.dieselnet.com/standards/cycles, 24.10.2010.
- [15] R. Kubalczik, M. Ebenhoch and H. Schneider. 7-Gang Doppelkupplungsgetriebe für sportliche Anwendungen. In *VDI Berichte 1943, Getriebe in Fahrzeugen 2006*, page 309...324, Düsseldorf, 2006. VDI Verlag.

Nonlinear Observers based on the Functional Mockup Interface with Applications to Electric Vehicles

Jonathan Brembeck, Martin Otter, Dirk Zimmer

German Aerospace Center (DLR) Oberpfaffenhofen, Institute of Robotics and Mechatronics
Münchner Strasse 20, D-82234 Wessling, Germany
jonathan.brembeck@dlr.de, martin.otter@dlr.de, dirk.zimmer@dlr.de

Abstract

At DLR, an innovative electric vehicle is being developed that requires advanced, nonlinear control systems for proper functioning. One central aspect is the use of nonlinear observers for several modules. A generic concept was developed and implemented in a prototype to automatically generate a nonlinear observer model in Modelica, given a continuous (usually nonlinear) Modelica model of the physical system to be observed. The approach is based on the Functional Mockup Interface (FMI), by exporting the model in FMI format and importing it again in a form that enables the application of different observer designs, like EKF and UKF nonlinear Kalman Filters. The approach is demonstrated at hand of an observer for the nonlinear battery model of the electric vehicle of DLR.

Keywords: FMI, FMU, Kalman Filter, EKF, UKF

1 Introduction

The ROboMObil (Figure 1, [Bre11]), a research platform for future electro mobility is developed at the DLR Institute for Robotics and Mechatronics. Its fully centralized control architecture enables highly innovative control strategies. For most of these methods, a good knowledge of all actuator states is required. Unfortunately, many of them cannot be measured directly and therefore have to be estimated. In [Eng10], a concept for one of the ROboMObil actuators was developed to implement recursive estimator algorithms in Modelica manually based on the Functional Mockup Interface [FMI10], [FMI11]. This approach is enhanced in this paper such that, at least in principle, every Modelica model can be automatically utilized in a nonlinear observer. In the

following sections, the utilized recursive state estimation algorithms are summarized, the implementation in Modelica is outlined, and a universal Phyton based [Phy10] FMI importer is presented. Finally, experimental results with the Lithium-Ion cells of the ROboMObil in combination with this new observer framework are demonstrated.



Figure 1: ROboMObil test drive

2 Recursive state estimation

In this chapter, the principle ideas of recursive state estimation are summarized, and its (historical) development leading to the Kalman Filter is outlined. In the second part, this algorithm is extended to nonlinear systems and finally the latest developments are sketched. Further background information, alternative formulations, and recent developments are provided in the standard book [Sim06] that is also the starting point for the following explanations.

2.1 Principles

At first, we consider an estimation of a *constant signal* on the basis of several noisy measurements. This *Weighted Least Squares Estimation* problem is well-known in system identification tasks (see, e.g., [Lju98]). Through the weighted formulation, the user can assign different levels of confidence to certain measurements (or observations). This feature is crucial for tuning Kalman Filters. The corresponding minimization problem is formulated as follows:

$$\begin{bmatrix} y_1 \\ \vdots \\ y_k \end{bmatrix} = \begin{bmatrix} H_{11} & \dots & H_{1n} \\ \vdots & \ddots & \vdots \\ H_{kn} & \dots & H_{kn} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} v_1 \\ \vdots \\ v_k \end{bmatrix} \quad (1)$$

$$E(v_i^2) = \sigma_i^2 \quad (i = 1, \dots, k)$$

$$x \in \mathbb{R}^n, y \in \mathbb{R}^k, v \in \mathbb{R}^k$$

The unknown vector x is constant and consists of n elements, y is a k -element noisy measurement vector and usually $k \gg n$. Each element of y - y_i - is a linear combination (H_{k*}) with the unknown vector x and the variance of the measurement noise of the i -th measurement v_i . The noise of each measurement is zero-mean and independent from each other, therefore the measurement covariance matrix is

$$R = (vv^T) = \text{diag}(\sigma_1^2, \dots, \sigma_k^2) \quad (2)$$

The residual

$$\epsilon_y = \underbrace{(Hx + v)}_{=y} - H\hat{x} \quad (3)$$

is the difference of all measured values y with the (unknown) x -vector minus the estimated vector \hat{y} that is computed from the estimated vector \hat{x} . The goal is to compute the estimated vector \hat{x} such that the weighted residual is as small as possible, i.e., to minimize the cost function J :

$$J = \frac{\epsilon_{y1}^2}{\sigma_1^2} + \dots + \frac{\epsilon_{yk}^2}{\sigma_k^2} \quad (4)$$

To minimize J , it is useful to compute the partial derivative with respect to the estimated \hat{x} vector and set it to zero. In this way, an optimal solution for \hat{x} can be calculated:

$$\frac{\partial J}{\partial \hat{x}} = 2 \cdot (-y^T R^{-1} H + \hat{x}^T H^T R^{-1} H) = 0 \quad (5)$$

$$\hat{x} = (H^T R^{-1} H)^{-1} H^T R^{-1} y$$

(5) requires that R is nonsingular and H has full rank. This is the “textbook” version of the algorithm. It is inefficient and numerically not reliable.

Alternatively, (4) can be formulated as:

$$J = \begin{bmatrix} \frac{\epsilon_{y1}}{\sigma_1} & \dots & \frac{\epsilon_{yk}}{\sigma_k} \end{bmatrix} \cdot \begin{bmatrix} \frac{\epsilon_{y1}}{\sigma_1} \\ \vdots \\ \frac{\epsilon_{yk}}{\sigma_k} \end{bmatrix} \quad (6)$$

To solve the following standard linear least squares problem that minimizes the Euclidian norm of the weighted residue vector:

$$\begin{aligned} \min_{\hat{x}} & \left\| \begin{bmatrix} \frac{\epsilon_{y1}}{\sigma_1} & \dots & \frac{\epsilon_{yk}}{\sigma_k} \end{bmatrix} \right\|_2 \\ & = \min_{\hat{x}} \|W(y - H\hat{x})\|^2 \\ & = \min_{\hat{x}} \|WH\hat{x} - Wy\|^2 \\ & = \min_{\hat{x}} \|A\hat{x} - b\|^2 \end{aligned} \quad (7)$$

$$W = \text{diag}(1/\sigma_1, \dots, 1/\sigma_k)$$

This minimization problem has a unique solution, if $A=WH$ has *full rank*. If A is *rank deficient*, an infinite number of solutions \hat{x} exists. The usual approach is to select from the infinite number of solutions the unique one that additionally minimizes the norm of the solution vector: $\|\hat{x}\|^2 \rightarrow \min$. Given $A=WH$ and $b = Wy$, this solution vector can be computed with the Modelica function `Modelica.Math.Matrices.leastSquares(...)` from the Modelica Standard Library which is a direct interface to the LAPACK function `DGELSX` [Lap99].

This function uses a QR decomposition of A with column pivoting together with a right multiplication of an orthogonal matrix Z to arrive at:

$$\min_{\hat{x}} \left\| \begin{bmatrix} Q_1 & Q_2 \end{bmatrix} \begin{bmatrix} U & 0 \\ 0 & 0 \end{bmatrix} ZP\hat{x} - b \right\|^2 \quad (8)$$

where Q and Z are orthogonal matrices, P is a permutation matrix, U is a regular, upper triangular matrix and the dimension of the quadratic matrix U is identical to the rank of A . Since the norm of a vector is invariant against orthogonal transformations, this equation can be transformed to:

$$\min_{\hat{x}} \left\| \begin{bmatrix} U & 0 \\ 0 & 0 \end{bmatrix} ZP\hat{x} - \begin{bmatrix} Q_1^T b \\ Q_2^T b \end{bmatrix} \right\|^2 \quad (9)$$

This is equivalent to

$$\min_{\hat{x}} \left\| \begin{bmatrix} U \\ 0 \end{bmatrix} \hat{x}_1 - \begin{bmatrix} Q_1^T b \\ Q_2^T b \end{bmatrix} \right\|^2, \hat{x} = ZP\hat{x} \quad (10)$$

from which the solution can be directly computed as (taking into account $b = Wy$):

$$\hat{x} = PZ^T U^{-1} Q_1^T W y \quad (11)$$

In the following, only textbook versions of algorithms will be shown, such as (5). Their implementation is, however, performed in an efficient and numerically reliable way, such as (11), where matrices R and H can be rank deficient.

The sketched approach, both (5) and (11), can be used for *offline estimation* with a predetermined number of measurements k .

In real-time applications, new measurements arrive in each sample period to improve the estimation. Using (11) would require a complete recalculation with $O(k^3)$ -flops. One approach could be to use a moving horizon and to forget the older measurements (still

costly). Another option is to reformulate the problem into a recursive form that is updated at every sample instant with the new measurements. A linear recursive estimator can be written in the following representation:

$$\begin{aligned} y_k &= H_k x + v_k \\ \hat{x}_k &= \hat{x}_{k-1} + K_k \cdot (y_k - H_k \hat{x}_{k-1}) \end{aligned} \quad (12)$$

We compute \hat{x}_k based on the estimation from the last time step \hat{x}_{k-1} and the information from the new measurement y_k . K_k is the estimator gain vector that weights the correction term $y_k - H_k \hat{x}_{k-1}$. Hence, we have to compute an optimal K_k in a recursive way. To this end, it is necessary to formulate another cost function that minimizes the covariance in a recursive way.

$$\frac{\partial J_k}{\partial K_k} = \frac{\partial \text{Tr} P_k}{\partial K_k} = 0 \quad (13)$$

$$P_k = (I - K_k H_k) P_{k-1} (I - K_k H_k)^T + K_k R_k K_k^T \quad (14)$$

$$K_k = P_{k-1} H_k^T \cdot (H_k P_{k-1} H_k^T + R_k)^{-1} \quad (15)$$

This results in a recursive formula to update the estimation of the unknown, but constant, vector x in every sample with the latest measurements, based only on the estimation from the last sample. Table 1 summarizes the whole algorithm.

Table 1: Recursive weighted least squares algorithm

Initialization $\hat{x}_0 = E(x)$ $P_0 = E[(x - \hat{x}_0)(x - \hat{x}_0)]$ For $k = 1, 2, \dots$ $y_k = H_k x + v_k$ $K_k = P_{k-1} H_k^T \cdot (H_k P_{k-1} H_k^T + R_k)^{-1}$ $\hat{x}_k = \hat{x}_{k-1} + K_k \cdot (y_k - H_k \hat{x}_{k-1})$ $P_k = (I - K_k H_k) P_{k-1} (I - K_k H_k)^T + K_k R_k K_k^T$

For many real-time control problems, it is more interesting to estimate the system states rather than some constant parameters. Therefore the *linear Kalman Filter* was developed in the 60's. It enables to estimate the system states of a linear discrete-time model in a recursive way. The fundamental assumption is that the system and the output equations are disturbed by white Gaussian noise. Both of these noise processes are regarded as uncorrelated with zero mean. This results in the following equations:

$$\begin{aligned} x_k &= F_{k-1} x_{k-1} + G_{k-1} u_{k-1} + w_{k-1} \\ y_k &= H_k x_k + v_k \\ E(w_k w_j^T) &= Q_k \delta_{k-j} \\ E(v_k v_j^T) &= R_k \delta_{k-j} \\ E(w_k v_j^T) &= 0 \end{aligned} \quad (16)$$

At this point, we introduce the principle of every Kalman Filter derivation (compare Figure 2). Subsequent to filter initialization, the first step in every sample is the a-priori estimation of the mean (system states) and the covariance (a gauge for the confidence in them). This is called the prediction step and all of the equations that are related with it contain a “-“ in the superscript.

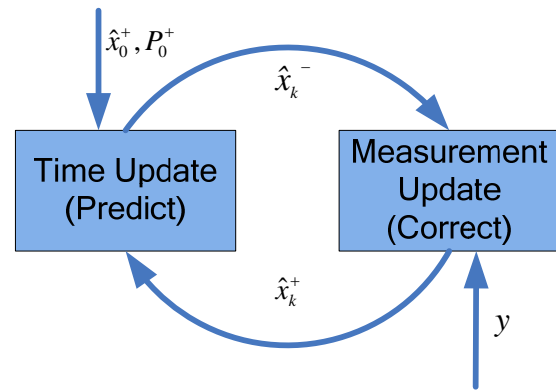


Figure 2: Principle of recursive Kalman filter.

This forms the basis for the calculation of the optimal Kalman gain that is used to correct the estimated state vector with the information from the actual measurements. Finally, the covariance matrix is updated. This is called the correction step. In the next sample, these values are used to restart again at the subsequent prediction step. The algorithm can be formulated as follows:

Table 2: Linear discrete Kalman Filter

Initialization $\hat{x}_0 = E(x_0)$ $P_0^+ = E[(x_0 - \hat{x}_0^+)(x_0 - \hat{x}_0^+)]$ For $k = 1, 2, \dots$ $\hat{x}_k^- = F_{k-1} \hat{x}_{k-1}^+ + G_{k-1} u_{k-1}$ $P_k^- = F_{k-1} P_{k-1}^+ F_{k-1}^T + Q$ $K_k = P_k^- H_k^T \cdot (H_k P_k^- H_k^T + R)^{-1}$ $\hat{x}_k^+ = \hat{x}_k^- + K_k \cdot (y_k - H_k \hat{x}_k^-)$ $P_k^+ = (I - K_k \cdot H_k) \cdot P_k^-$
--

To determine the relationship between the *Kalman Filter* and *recursive weighted least squares*, we should have a closer look at Table 2. The matrix $Q(w w^T) = \text{diag}(\sigma_{w1}^2, \dots, \sigma_{wn}^2)$ represents the covariance of the system states (w denotes the variance of the system states). Its entries represent the confidence in the a-priori estimation and can be tuned by the application engineer. Large values represent high uncertainty (probably due to an imprecise model), whereas small values indicate good trust. The second tuning matrix R represents the confidence in the actual measure-

ments. Its effect resembles our first estimation problem (eq. (1) to (5)). Furthermore, it can be shown that if x_k is a constant vector then $F_k = I$, $Q_k = 0$ and $u_k = 0$. In this case, the *Linear discrete Kalman Filter algorithm* (Table 2) reduces to the *recursive weighted least squares algorithm* (Table 1). This property is often exploited in the formulation of parameter estimation problems using *Kalman Filter algorithms*.

2.2 Nonlinear Kalman Filter Algorithms

So far, we have discussed estimation problems for linear discrete systems. This is generalized to nonlinear systems starting from a continuous-time representation in state space form:

$$\begin{aligned} \dot{x} &= f(x, u) \\ y &= g(x) \end{aligned} \quad (17)$$

In section 3, it is sketched how such a model description can be generated from a Modelica model for use in a *nonlinear Kalman Filter* using the Functional Mockup Interface. In this way, it is possible to formulate the synthesis models for the prediction step (see Figure 2) with Modelica, even in implicit representation, and shift all tedious tasks to the *Nonlinear Observer* framework. This avoids calculus mistakes and allows us to put the main focus on the design of the algorithms.

In Table 3, the widely used extension of the *discrete linear Kalman Filter* to the *discrete nonlinear Kalman Filter with additive noise* is presented. The dynamic system is represented as follows:

$$\begin{aligned} x_k &= f_{k-1}(x_{k-1}, u_{k-1}) + w_{k-1} \\ y_k &= h_k(x_k) + v_k \\ w_k &\cong (0, Q_k) \\ v_k &\cong (0, R_k) \end{aligned} \quad (18)$$

The algorithm is very similar to a purely linear one. To handle the nonlinearity, the system is linearized around the last estimation point using a Taylor Series Expansion up to the first term. This can be performed numerically by the use of a forward difference formula.

Table 3: Extended Kalman Filter Algorithm

Initialization $\hat{x}_0 = E(x_0)$ $P_0^+ = E[(x_0 - \hat{x}_0^+)(x_0 - \hat{x}_0^+)^T]$ For $k = 1, 2, \dots$ $\hat{x}_k^- = f_{k-1}(\hat{x}_{k-1}^+, u_{k-1})$ $P_k^- = F_{k-1} P_{k-1}^+ F_{k-1}^T + Q$ where $F_{k-1} = \left. \frac{\partial f_{k-1}}{\partial x} \right _{\hat{x}_{k-1}^+}$ $K_k = P_k^- H_k^T \cdot (H_k P_k^- H_k^T + R)^{-1}$ where $H_k = \left. \frac{\partial h_k}{\partial x} \right _{\hat{x}_k^-}$ $\hat{x}_k^+ = \hat{x}_k^- + K_k \cdot (y_k - h_k(\hat{x}_k^-))$ $P_k^+ = (I - K_k \cdot H_k) \cdot P_k^-$

Since we have a nonlinear continuous-time system representation, we have to linearize and discretize our system at every sample instant. Discretization means to integrate the system in the prediction step from the last sample instant to the new one, e.g. with the *Trapezoidal* or the *Runge-Kutta 4* integration method. The transition matrix F_{k-1} is calculated by an analytic derivation of the system state Jacobian. An alternative is the numerical calculation with, e.g., a forward difference formula:

$$\text{For } i = 1, 2, \dots, n$$

$$J_{\hat{x}_{k-1}^+}^{[:,i]} = \frac{f(\hat{x}_{k-1}^+ + h \cdot E(:, i), u) - f(\hat{x}_{k-1}^+, u)}{h} \quad (19)$$

The transition matrix can be computed with function `Modelica.Math.Matrices.exp` from the Modelica Standard Library resulting in:

$$F_{k-1} = e^{(J_{\hat{x}_{k-1}^+} \cdot T_s)} \quad (20)$$

The same procedure is necessary to calculate the output Jacobian H_k . Using this method, it is possible to use a nonlinear continuous-time system within the *discrete nonlinear Kalman Filter* algorithm.

The discussed EKF algorithm is widely used in many applications. However, it often gives unsatisfactory results or even does not converge if the system nonlinearities are severe because the linearization causes a propagation of the mean and covariance that is only valid up to the first order. The following section sketches the principles of the *Unscented Kalman Filter* (UKF) and its advantages in nonlinear state estimation.

2.3 Unscented Kalman Filter

In order to achieve higher accuracy, the UKF calculates the means and covariances from disturbed state vectors, called sigma points, by using the nonlinear system description. As one side effect, the Jacobians of $f(x)$ and $h(x)$ are no longer needed. See [Mer04] for more detailed information. The structure of the equation set, containing prediction and update, is similar to the EKF. However, the calculation of the covariances requires to integrate the nonlinear system $2n + 1$ times from the last to the actual time instant and is therefore computationally costly. The symmetry of all the involved matrices is fully exploited to reduce computational costs. An additional reduction of computational effort is achieved with the Square Root UKF (SR-UKF).

2.4 Square Root Unscented Kalman

The equations of the SR-UKF are identical to the UKF, but the structure is utilized during the evaluation: Although the covariance matrix P_k and the predicted covariance matrix P_k^- are uniquely defined by their Ckolesky factors $\sqrt{P_k}$ and $\sqrt{P_k^-}$ respectively, with UKF the covariance matrices are calculated at each step. Furthermore, the sigma points X_k can be computed with the Cholesky factor $\sqrt{P_k}$, and the updated sigma points of the measurement update with the Cholesky factor $\sqrt{P_k^-}$ without using the covariance matrices. Moreover, the gain matrix K_k is determined as solution of the linear equation system

$$K_k \cdot P_{y_k y_k} = P_{x_k y_k} \quad (21)$$

that can be more efficiently solved by utilizing again the Cholesky factorization. In the SR-UKF implementation, the Cholesky factors are propagated directly and the refactorization of the covariance matrices is avoided [Mer01b].

The EKF, UKF, and SR-UKF algorithms are implemented as Modelica functions using LAPACK for core numerical computations. Implementation details of the numerical algorithms will be provided in an upcoming publication by Marcus Baur.

3 Nonlinear Observers in Modelica

In this section a prototype implementation is sketched for applying the nonlinear observers from the previous section to Modelica models. The goal is to start from a given (continuous, usually nonlinear)

Modelica model and provide automatically a nonlinear observer for this model in form of a sampled data system.

This task cannot be performed directly, because Modelica has no means to discretize a continuous model and to solve this discretized model with a user-defined method (= integration + update of the next state according to the observer equations).

Note, it is insufficient to simply integrate the nonlinear models from the last to the new sample instant (which could be achieved by using the “mapping” annotation introduced in Modelica 3.1). Instead, the extended Kalman filter additionally requires linearizing the model around the sample time and using it together with the solution of the integration to compute a new estimation of the state that is utilized in the next step. On the other hand, the unscented Kalman filter requires integrating the model several times with disturbed states from the last to the new sample instant.

To summarize, there is no way to describe a nonlinear observer completely in Modelica and it is also very unlikely that the Modelica language is extended so that this becomes possible.

The basic approach is to export the Modelica model in the FMI-format (see section 3.1), import it again in Modelica and during import call the FMI-functions in such a way that the model is discretized and utilized in a nonlinear observer algorithm.

3.1 Functional Mockup Interface

The Functional Mock-up Interface (FMI) for Model Exchange [FMI10], [FM11] was developed in the MODELISAR project to standardize the exchange of dynamic models between tools. This interface is supported already by Dymola, SimulationX, JModelica.org, Silver and Simulink¹. Other tools are planning to support it as well.

The goal of the FMI is to describe input/output blocks of dynamic systems defined by differential, algebraic and discrete equations and to provide an interface to evaluate these equations as needed in different simulation environments, as well as in embedded control systems, with explicit or implicit integrators and fixed or variable step-size. Some details of the type of systems that can be handled are shown in Figure 3 (from [FMI10]).

¹ Dymola 7.4 can export Simulink models in FMI-format via Realtime-Workshop of MathWorks.

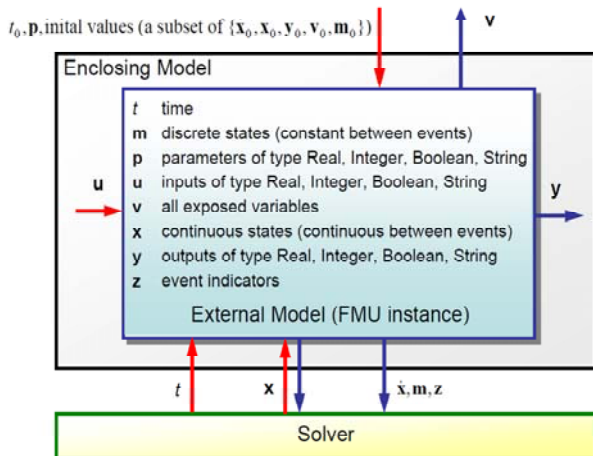


Figure 3: FMI for model exchange.

The interface consists of (a) a small set of standardized “C-functions” to evaluate the model equations and (b) an XML-file that contains all information that is not needed during execution, such as the variable definitions. Every variable has a handle (a 32 bit Integer) that is used to identify the variable in the C-function calls. The source and/or object code of the C-functions, as well as the XML-file and optionally other files, are stored in a zip-file with the extension “.fmu” for “Functional Mockup Unit”.

In order to implement nonlinear observers for Modelica models, the corresponding model has to be exported by one of the tools in FMI format. In a subsequent step, it has to be imported again. Unfortunately, a standard FMU-import as supported by Dymola and other tools cannot be used, because these interfaces import a model as continuous model, if it was exported as continuous model. For this reason, a new FMU-import method was implemented (see section 3.3). From the Modelica perspective, it was necessary to use the new feature of “functions as input argument to functions”, as introduced in the Modelica Language 3.2. This feature is currently only supported in Dymola 7.5 Beta. So we used this Dymola version for the prototype implementation.

3.2 FMU Definition in Modelica

The key point is that all FMI-functions of an imported FMU need to be available for design methods in Modelica. This is achieved in the following way:

1. A FMU (so a model exported by a Modelica tool) is mapped to a replaceable package consisting of (a) an external object that holds the “internal memory” of the model, (b) external functions that call the FMU functions, and (c) a Modelica model to instantiate and initialize the external object optionally defining new values for the pa-

rameters. This is a similar approach as used for media from the Modelica.Media package.

2. Design functions, such as computing the new estimated state of a model, are implemented in a model independent way. This is achieved by providing functions (as input arguments) that compute the needed information from a model. Concrete implementations of these functions are provided for FMUs.

Here is a more detailed sketch of this approach:

Package PartialFmiFunctions defines the interfaces to all FMI functions:

```

partial package PartialFmiFunctions
constant Integer nx=1 "# of states";
constant Integer nu=1 "# of inputs";
constant Integer ny=1 "# of outputs";
constant Integer id_u[nu]"Input handles";
constant Integer id_y[ny]"Output handles";

replaceable partial class FmiInstance
extends ExternalObject;
replaceable partial function constructor
input String instanceName;
input Boolean loggingOn;
output FmiInstance fmi;
end constructor;
replaceable partial function destructor
input FmiInstance fmi;
end destructor;
end FmiInstance;

replaceable partial function fmiSetTime
input FmiInstance fmi
input Real ti;
input Real preAvail;
output Real postAvail = preAvail;
end fmiSetTime;

replaceable partial function
fmiSetContinuousStates
input FmiInstance fmi;
input Real x[:];
input Real preAvail;
output Real postAvail= preAvail;
end fmiSetContinuousStates;

...
end PartialFmiFunctions;

```

It is important that the dimensions of the input, output and state vectors, as well as the vector of handles for the input variables (id_u) and for the output variables (id_y) are available in the package as constants, since they are needed later by the specialized functions for the design models.

Importing an FMU means to *generate* a FMU specific Modelica *package* of the form (below: <MODEL> is the name of the FMU):

```

package <MODEL>_fmu
model Model
// Define parameters of the FMU
// Define inputs, outputs of the FMU
// initialize FMU

```

```

parameter String name = "<MODEL>"
Functions.FmiInstance fmi=
    Functions.FmiInstance(name);
...
end Model;

package Functions
extends PartialFmiFunctions(
    nx=4,
    nu=1,
    ny=2,
    id_u={352321536},
    id_y={335544320,335544321});

redeclare class FmiInstance
extends ExternalObject;
function constructor
    input String instanceName;
    input Boolean loggingOn;
    output FmiInstance fmi;
    external "C" fmi = <MODEL_init>
        (instanceName, loggingOn);
end constructor;
function destructor
    input FmiInstance fmi;
    external "C" <MODEL_close>(fmi);
end destructor;
end FmiInstance;

redeclare function extends fmiSetTime
    external "C"
        <MODEL_fmiSetTime>(fmi, ti);
end fmiSetTime;

...
end Functions;
end <MODEL>_fmu;

```

The imported FMU is now available as a package that contains a model to initialize the FMU and a set of functions to operate on the initialized FMU.

Up to this stage, the code is completely independent from the design that shall be carried out, and the generated FMU package can be utilized for all kinds of design tasks. For every specific design, like an UKF observer, a model has to be implemented that has the following basic structure:

```

model UKF_FMI "Unscented Kalman filter"
import C =
    Modelica.LinearSystems2.Controller;
import I = Modelica.Blocks.Interfaces;
extends C.Interfaces.PartialDiscreteBlock
    (initType = C.Types.Init.InitialState);

replaceable package FmiFunctions =
    PartialFmiFunctions;
constant Integer nx = FmiFunctions.nx;
constant Integer ny = FmiFunctions.ny;
constant Integer nu = FmiFunctions.nu;
parameter Real Q[nx,nx]=identity(nx);
parameter Real G[nx, nx];
parameter Real R[ny, ny];
parameter Real P_init[nx,nx];
parameter Real x_init[nx] "Initial states";

input FmiFunctions.FmiInstance fmi;
I.RealInput u[nu] "Input u";
I.RealInput y_measure[ny] "Measured y";
I.RealOutput x_est[nx] "Estimated x";
I.RealOutput y_est[ny] "Estimated y";

```

```

Real time_;
Real P[nx,nx] "Error covariance matrix";
Real K[nx,ny] "Kalman filter gain matrix";
...
protected
    outer C.SampleClock sampleClock ;
initial algorithm
    x_est :=FmiFunctions.fmiGetContinuousStates
        (fmi,nx,1);
    P := P_init;
    time_ := 0;
algorithm
    when sampleTrigger then
        (x_est,y_est,P,K) := UKF(
            function fFMI(fmi=fmi),
            function hFMI(fmi=fmi),
            pre(x_est),pre(u),y_measure, ...);
        time_ :=time_ + sampleClock.sampleTime;
        FmiFunctions.fmiSetTime(fmi,time_,1);
        FmiFunctions.fmiCompletedStep(fmi,3);
    end when;
end UKF_FMI;

```

The UKF_FMI design model uses the PartialFmiFunctions as replaceable package to get access to the FMU functions of the model (in the same way as a medium is used in a fluid model), as well as an instance of the external object in this package (FmiInstance) to hold the internal memory of the FMU.

All data that the user has to provide for this design method is provided via parameters and input signals. The central code consists basically of a periodically evaluated when-clause where in every sample interval the UKF design function is called. This design function, here: UKF(...), is generic and does not depend on FMI. In case of the UKF, the design function requires two functions as inputs: fFMI(.) and hFMI(..). In model UKF_FMI above, these (generic) functions will internally call FMI functions, and therefore the handle to the FMU external object is provided as additional argument via a “function partial application”.

Function “fFMI” integrates the FMU over one sample period, whereas “hFMI” computes the output signals at the new sample time. For example, fFMI is implemented as:

```

function fFMI
    input FmiFunctions.FmiInstance fmi;
    input Real u[:] "Input at instant k";
    input Real x[:] "State at instant k";
    input Modelica.SIunits.Time Ts;
    output Real x_new[size(x, 1)]
        "Predicted x at k+1";
algorithm
    FmiFunctions.fmiSetReal
        (fmi, FmiFunctions.id_u, u, 1);
    x_new := RkFix4(fmi,Ts,x);
end fFMI;

```

With “fmiSetReal”; the input values are set and with function “RkFix4” the FMU is integrated from the previous to the next sample instant using a Runge-Kutta method of order 4 with a fixed step size. The

design function “UKF” finally is an implementation of the algorithm sketched in section 2 using LAPACK [Lap99] for its numerical part.

All pieces can now be assembled together. Assume for example, that a crane model is exported as FMU and that the importer of section 3.3 generated the package “Crane_fm” according to “<MODEL_fm>” from above. Then the code for an UKF observer for this model has basically the following structure:

```

model CraneObserver
// FMU instance
Crane_fm.Model CraneFMU(...);

// Unscented Kalman Filter
UKF_FMI UKF(
  fmi = CraneFMU.fmi,
  redeclare package FmiFunctions =
    Crane_fm.Functions,
  ...)

// Connect input and measurement signals
// to model UKF
end CraneObserver;
    
```

In the first statement an instance of the FMU model is generated. In the second statement, the model of the unscented Kalman filter is used and the FMU instance as well as the FMU functions are provided as arguments, besides Kalman specific settings.

3.3 FMU import using Python

To support the reimport of a FMU into a Modelica model in the specific form of section 3.2, a tool box has been developed in Python 3 [Phy10]. It consists of a library of Python classes and a set of scripts representing the end-user applications. Using this tool-box, a developer can easily create its own re-import functionality for FMUs, specially tailored to fit his or her set of demands. The result of the final Python script is Modelica package <MODEL>_fmu from the last section representing the imported FMU. The required input consists in the XML-file that is extracted from the FMU zip-file, optionally additional text-input by the user, and most important a template file, see Figure 4. This template file consists of a Modelica model file that contains mark-up elements to be replaced by the Python Script.

The template file for FMUs for nonlinear observers resembles the structure of package “<MODEL>_fmu” sketched in section 3.2.

Using the Python tool-box, FMUs can be re-imported into Modelica in a very flexible way suiting a broad set of potential future applications.

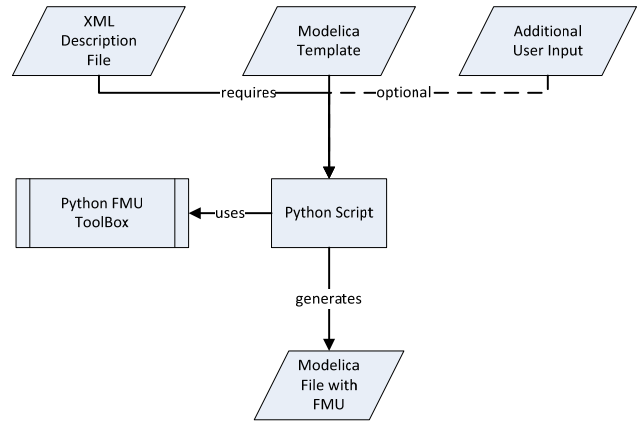


Figure 4: Processing Scheme for the Python-based FMU Re-import

4 Example SOC estimation

Subsequently, the observer framework is demonstrated in an application from the development of the ROboMObil. The battery model introduced in [Bre11b] is used as the synthesis model for the FMU-Export. The observer scheme is shown in Figure 5.

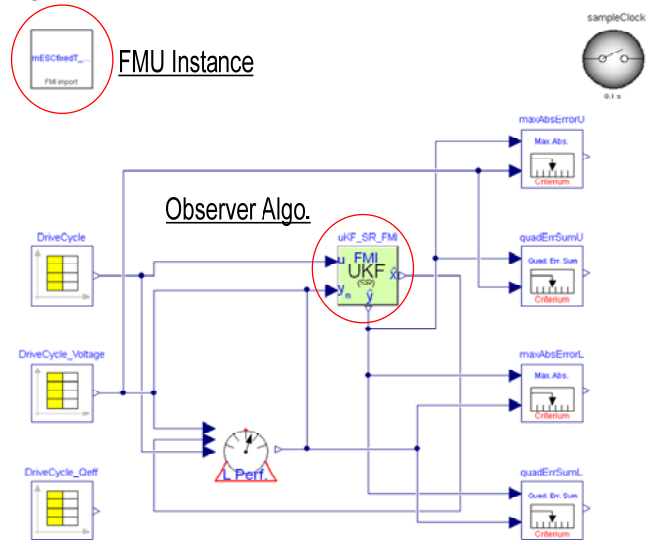


Figure 5: FMU based observer setup

In the top left corner of the model a FMU instance block is placed. The free parameters of the imported model can be tuned here before simulation. So it is possible to modify system parameters, i.e. due to changed conditions in the experiment, without the necessity of repeating the importing procedure. With these parameters and the system equations, the FMU instance calculates the initial states of the prediction model and instantiates the FMI object.

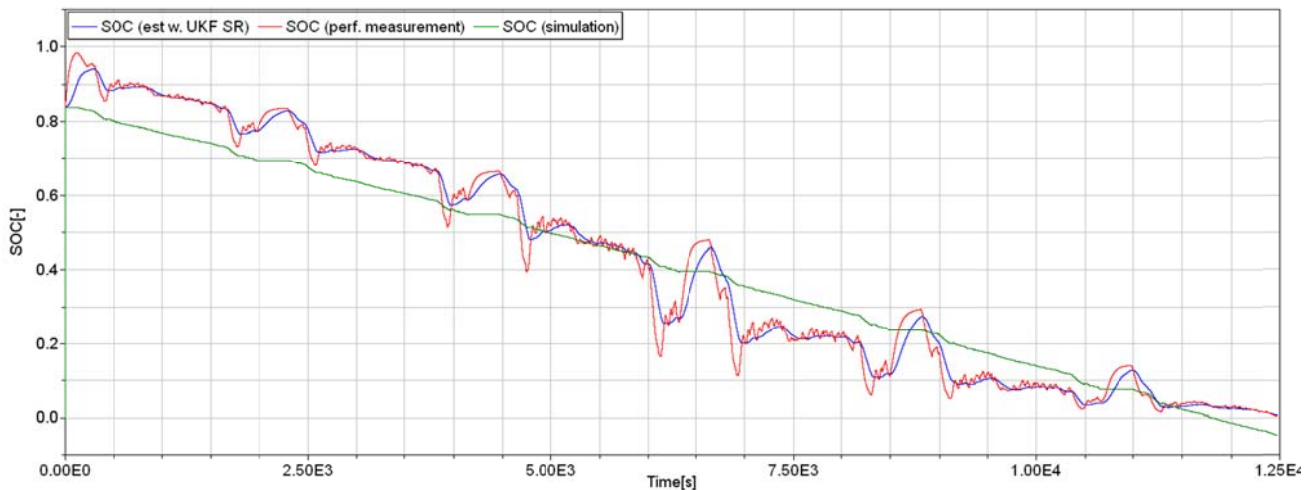


Figure 6: Comparison of observer vs. model based SOC characteristics

The pointer to this instance is passed to the observer algorithm. This can be done via the parameter dialog. In our case we have chosen an *UnscentedKalman-Filter* using *SquareRoot* matrix calculation. Moreover the user has to tune the free filter parameters like the covariance matrix or the sigma point spread. This has to be performed individually for every application, manually or by offline optimization methods ([Bre11b]).

In this example we like to estimate the *StateOfCharge* of a LiIon battery cell. The input u of the battery model is the measured current, while the model output vector y_m is the cell voltage and the noisy SOC which is calculated via the method of perfect measurements (c.f. Figure 5, component with description text “L Perf.”, where L is the state variable for the SOC).

For experiment data we use a FTP75 driving cycle which is simulated with the ROboMObil energetic model ([Eng10]). The calculated electric power demand of the actuators is converted to the current demand of one cell. This is used as current demand to the single cell test bench (Figure 7). The voltage at the cell terminals, the surface temperature and the effective current flow are recorded during this test. Finally they are used as input and measurement data of the experiment setup (Figure 5 bottom left).

In Figure 6 we have presented the experiment results and benefits of using a model based recursive observer in real-time applications. The red curve shows the SOC characteristic calculated via the perfect measurement.

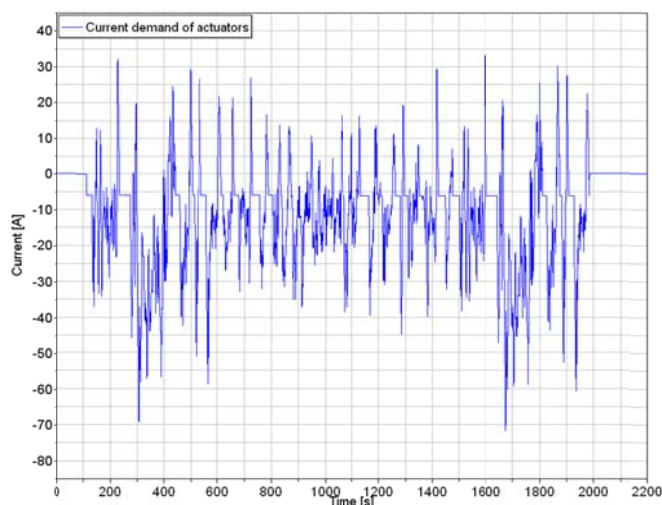


Figure 7: Current demand from ROboMObil in FTP75 drive cycle

It is, despite of signal pre-filtering, very erratic and noisy. This characteristic is qualitatively correct, especially in comparison to the green curve, which represents the output of a pure model simulation without observer correction. The pure simulation causes a SOC that is less than zero at the end of the simulation which is physically impossible (c.f. Figure 7, bottom right). In car applications, this would mean that the SOC display would show incorrect information. In this case it would not be possible to drive on, although the battery is not exhausted yet. Through our estimation algorithm, we get a better and smoother estimation of the SOC that converges to zero (blue curve) at the end. Due to the efficient code provided by the FMI interface, this test runs with a real time factor greater than 100 on standard desktop systems. Thereby, it is possible to implement this observer on embedded or rapid prototyping controllers within the ROboMObil.

5 Conclusions and future work

We have demonstrated a way to develop a framework for generic observer design. The algorithm part is completely separated from the synthesis model. This could be achieved by the use of the FMI reim- port mimic and the new possibilities of Modelica 3.2 to pass functions as arguments to functions. The pre- sented example of a battery state estimation and its results make us confident that this framework can be used for many control system tasks in the future, es- pecially in the ROboMObil project. Furthermore, the estimation algorithms will be extended to handle constraints in a recursive way, [Sim09] [Kan08] and to take “out of sequence measurements” into ac- count, [Lar98] [Mer04].

6 Acknowledgement

The financial support of DLR by BMBF (Förderkennzeichen: 01IS08002) for this work with- in the ITEA2 project MODELISAR (http://www.itea2.org/public/project_leaflets/MODELISAR_profile_oct-08.pdf) is highly appreciated.

References

- [And04] André M. (2004): **The ARTEMIS European driving cycles for measuring car pollutant emissions**. Science of The Total Environment, 334-335, pp. 73-84.
- [Bre11] Brembeck J., Ho L. M., Schaub A., Satzger C., Hirzinger P. G. (2011): **ROMO – the robotic electric vehicle**. IAVSD, Aug. 14-19, accepted for publication.
- [Bre11b] Brembeck J., Wielgos S. (2011): **A real time capable battery model (mESC) for electro mobility applications using optimal estimation methods**. Modelica'2011 Conference, March 20-22.
- [Eng10] Engst C. (2010): **Object-Oriented Modelling and Real-Time Simulation of an Electric Vehicle in Modelica**. Master Thesis, Technische Universität München, Lehrstuhl für elektrische Antriebssysteme und Leistungselektronik. Supervisors: J. Brembeck, M. Otter, R. Kennel.
- [FMI10] **The Functional Mock-up Interface for Model Exchange, Version 1.0** (2010). ITEA2 MODELISAR Project. Download: www.functional-mockup-interface.org/fmi.html
- [FMI11] Blochwitz T., Otter M., Arnold M., Bausch C, Clauß C., Elmqvist H., Junghanns A., Mauss J., Monteiro M., Neidhold T., Neumerkel D., Olsson H., Peetz J.-V., Wolf S. (2011): **The Functional Mockup Interface for Tool independent Exchange of Simulation Models**. Modelica'2011 Conference, March 20-22.
- [Kan08] Kandepu R., Imsland L., Foss B. (2008): **Constrained state estimation using the Unscented Kalman Filter**. Proceedings of the 16th Mediterranean Conference on Control and Automation, pp. 1453-1458, June 25-27.
- [Lap99] Anderson E., Bai Z., Bischof C., Blackford S., Demmel J., Dongarra J., Du Croz J., Greenbaum A., Hammarling S., McKenney A., Sorensen D. (1999): **Lapack Users' Guide**. Third Edition, SIAM. Download: <http://www.netlib.org/lapack>
- [Lar98] Larsen T. D., Andersen N. A., Ravn O., Poulsen N. K. (1998): **Incorporation of time delayed measurements in a discrete-time Kalman filter**. Proceedings of the 37th IEEE Conference on Decision and Control, vol. 4, pp. 3972-3977, Dec. 16-18.
- [Lju98] Ljung, L. (1998): **System Identification: Theory for the User**. Prentice Hall, 2nd Edition.
- [Mer01] Merwe R. V., Wan E. (2001): **The square-root unscented Kalman filter for state and parameter-estimation**. Proceedings of the International Conference on Acoustics, Speech, and Signal Processing vol. 6, pp. 3461-3464.
- [Mer04] Merwe R. V., Wan E., Julier S. (2004): **Sigma-Point Kalman Filters for Nonlinear Estimation and Sensor-Fusion: Applications to Integrated Navigation**. Proceedings of the AIAA Guidance Navigation & Control Conference, Providence, RI, Aug 2004. Download: http://www.csee.ogi.edu/~rudmerwe/pubs/pdf/vanderMerwe_GNC2004.pdf
- [Mod10] Modelica Association (2010): **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>.
- [Phy10] **Python 3.1.2- Documentation (2010)**: Download: <http://docs.python.org/py3k/>
- [Sim06] Simon D. (2006): **Optimal State Estimation: Kalman, H Infinity, and Nonlinear Approaches**. John Wiley & Sons Inc..
- [Sim09] Simon D. (2010): **Kalman filtering with state constraints: a survey of linear and nonlinear algorithms**. IET Control Theory & Applications, vol. 4, no. 8, pp. 1303-1318, Aug..

Using the Functional Mockup Interface as an Intermediate Format in AUTOSAR Software Component Development

Bernhard Thiele^{*},Dan Henriksson⁺^{*}German Aerospace Centre (DLR), Institute for Robotics and Mechatronics, Germany⁺Dassault Systèmes AB, Ideon Science Park, Lund, Sweden

Bernhard.Thiele@dlr.de, Dan.Henriksson@3ds.com

Abstract

This paper shows how the recently developed Functional Mockup Interface (FMI) standard for model exchange can be utilized in the context of AUTOSAR software component (SW-C) development. Automatic transformations between the XML schemas of the two standards are utilized to convert FMI models to AUTOSAR. An application example is demonstrated, where a Modelica controller is exported through FMI, converted to an AUTOSAR SW-C and then imported into an AUTOSAR tool. The presented approach, with FMI as an intermediate format, should be an attractive alternative to providing full-fledged AUTOSAR SW-C export.

Keywords: FMI; AUTOSAR; model-based design; embedded software

1 Introduction

During the last two years, an open standard for exchange of simulation models, the Functional Mockup Interface (FMI), has been developed within the European ITEA2 research project MODELISAR. This standardized interface supports exchange of models that are described by differential, algebraic and discrete equations with time-, state- and step-events. The first official version, 1.0, of this standard was released on January 26, 2010.

Apart from the obvious improvements for model exchange between different tools and vendors, the interface is also well suited, and designed, for software components in embedded control systems. Since one of the major industrial driving forces behind the MODELISAR project is within the automotive industry, interoperability of the lightweight FMI with the comprehensive AUTOSAR standard for automotive E/E applications is of high interest. This paper examines the applicability of using FMI within

an AUTOSAR-based software component development process.

The paper is organized as follows. Details of the FMI and AUTOSAR standards are given in Sections 2 and 3, respectively. A mapping and conversion between FMI and AUTOSAR is then described in Section 4. An example application involving the Dymola [1] and AUTOSAR Builder [2] tools are presented in Section 5. Finally, Section 6 gives the conclusions.

2 Functional Mockup Interface

Integration of components delivered by many different suppliers is a common task in modern product engineering. To reduce costs, control complexity, and accelerate development it is desirable to allow this integration task to be done using a virtual representation of the product, i.e., to build a digital mockup. Besides spatial integration of the different components in a CAD tool, it is also required to let the dynamic behavior of the product to be predicted and checked by means of (physical) simulation.

Very often suppliers already have dynamic system models of their particular component, developed within their preferred simulation tool. However, integrating the various component models (possibly each developed with a different simulation tool) into an overall system model for joint simulation has proven to be a rather difficult, time-consuming, and numerically fragile undertaking.

The intention of the Functional Mockup Interface (FMI) is that dynamic system models from different tool vendors can be coupled together to form an overall system model with minimal effort and high numerical quality. To achieve that goal, the FMI defines an open interface that needs to be implemented by tools in order to import or export FMI system models. In FMI terminology a system model that implements the interface defined by the FMI specification is called a Functional Mockup Unit (FMU).

Figure 1 from the MODELISAR project profile description shows a use-case from an automotive OEMs perspective.

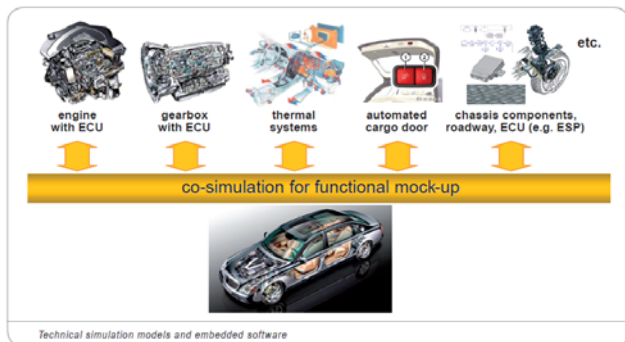


Figure 1: A functional mock-up of a vehicle consisting of several coupled Functional Mock-up Units (source: www.itea2.org)

3 Introduction to AUTOSAR

AUTOSAR is an automotive standard, which aims to decouple hardware and software and to separate communication from function. It achieves that by introducing several layers of abstraction with standardized interfaces.

The development partnership AUTOSAR (<http://www.autosar.org>) has released version 4.0 of the AUTOSAR standard in December 2009. However, since most commercially available tools to this date not yet support the 4.0 release, the following discussion concentrates on the 3.1 release of the standard.

The actual functional behavior (e.g. a model-based control algorithm) is encapsulated in AUTOSAR Software Components (SW-Cs). These components are decoupled through standardized interfaces from specific characteristics of Electronic Control Units (ECUs) and the given communication mechanism (e.g., automotive buses like CAN, FlexRay, LIN or inter-process communication if several software components interact on the same ECU).

The benefit of this decoupling is that the software components can be moved without adaption between different ECUs. The interconnections between the software components are handled by the Virtual Functional Bus (VFB). The VFB is the sum of all communication mechanisms and essential interfaces to the basic (hardware-dependent) software provided by AUTOSAR on an abstract level to software components (see Figure 2).

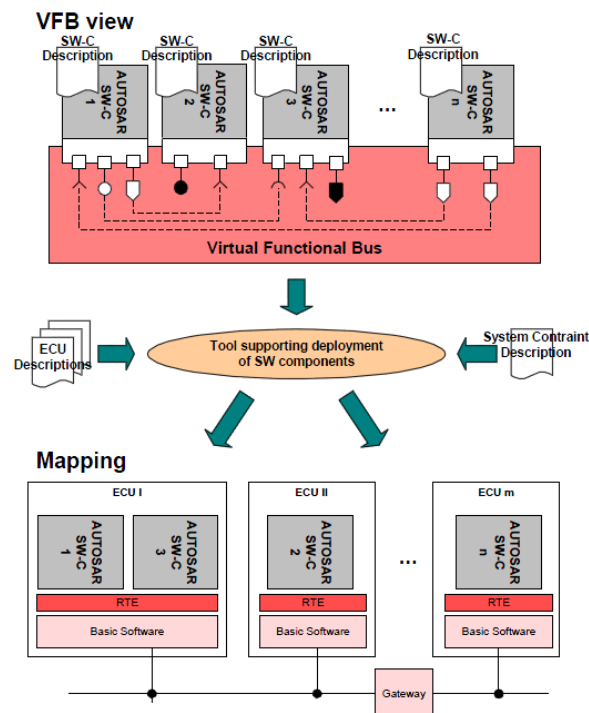


Figure 2: Basic AUTOSAR approach for configuration of an AUTOSAR system (source: [7], p. 9).

The mapping of the software components to the physical ECUs, as well as the mapping of the software component’s communication ports to the physical communication mechanism (e.g., CAN, FlexRay, LIN, or shared memory) is provided in a later configuration step. This allows starting the development of the logical software functions independently from the decision of the target platform (following the concept of separation of *logical system architecture* from the *technical system architecture* [3] [4]).

After that configuration, an AUTOSAR tool can deduce what software/communication functionality is required on a particular ECU and will be able to generate the needed source code for the particular ECU (target platform). This means that the abstract communication connections modeled on the VFB level are transformed to concrete communication connections on the ECUs. The software layer that provides the VFB communication services for the SW-C is called AUTOSAR Runtime Environment (RTE) and needs to be generated by the tool for every ECU.

4 FMI to AUTOSAR Software Component conversion

The development of the FMI is primarily intended to provide a standardized exchange format for *physical*

simulation models [1]. Nevertheless the intention to use that standard also for software components in embedded control systems is already stated in the abstract of [6].

Compared to AUTOSAR, the FMI standard is much smaller and more straightforward, and support of the FMI standard is a more manageable task¹. Thus, a conversion from FMI to AUTOSAR SW-Cs could be a cost effective alternative to providing dedicated AUTOSAR code generators (especially if support for FMI is already available or planned).

4.1 Establishing a relation between FMI and AUTOSAR software component specification methodology

Both FMI and AUTOSAR use XML documents for capturing the information about the (software) model (see [6] and [7]). In each case, the structure of the XML documents is defined in an associated XML schema [8]. A notable difference is that the AUTOSAR 3.1 schema occupies about 1000KB, while the FMI 1.0 schema is limited to about 25.5KB.

Mapping between different XML schemas is a common IT task and dedicated standards and tools are readily available. The Altova MapForce [9] program is a tool that allows defining mappings between XML schemas in a graphical manner. Figure 3 shows an excerpt of a mapping from FMI to AUTOSAR 3.1 developed in MapForce which was utilized in the first prototype mapping².

There is no univocal relation between FMI and AUTOSAR elements. Therefore design decisions about the available alternatives need to be made.

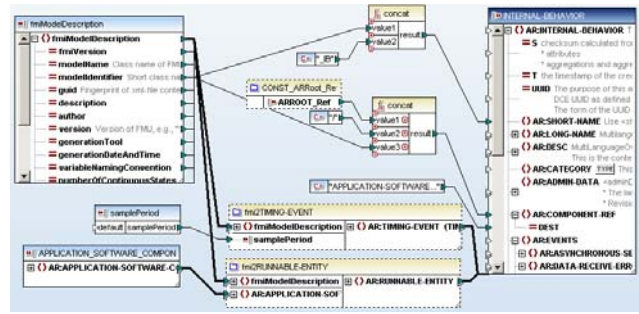


Figure 3: Excerpt of the mapping between the FMI and AUTOSAR schema.

4.2 Mapping FMI inputs/outputs to AUTOSAR SW-Cs Ports

The interaction between AUTOSAR Software Components and other parts of the system (including other AUTOSAR Software Components) is realized over a set of ports with standardized interfaces. Figure 4 shows the graphical representation of an AUTOSAR SW-C with different ports at its interface boundary.

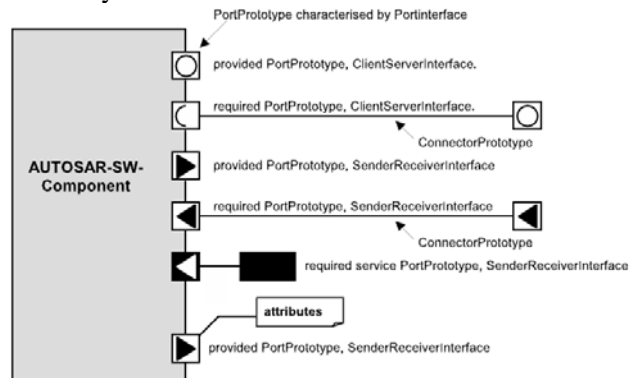


Figure 4: Graphical representation of software-components in AUTOSAR (source: [12], p. 20).

There are basically three kinds of port interfaces supported by AUTOSAR:

- Client-server: The server is the provider of operations and several clients can invoke those operations.
- Sender-receiver: A sender distributes information to one or several receivers, or one receiver gets information (events) from several senders.
- Calibration: Using or providing (static) calibration data

A port can either be a “*PPort*” or an “*RPort*”. A “*PPort*” provides the elements defined in a port interface. An “*RPort*” requires the elements defined in a port interface.

The FMI standard collects all visible/accessible variables within one central data structure (in the “*ModelVariables*” element). That element contains a

¹ In particular the import of AUTOSAR SW-Cs is much more complex, than that of importing an FMU. The reason for this is the great flexibility of the AUTOSAR standard to define SW-Cs, which needs to be managed by an importer. So using FMI as interchange format for embedded software components could also facilitate the exchange of embedded software.

² In later versions the mapping in MapForce was dropped in favor of a mapping developed in Scala [10] and Java utilizing auto-generated XML data bindings from the Altova XMLSpy tool [11]. The reason for that was the perceived need for more flexible language expressiveness as the mapping became more complex.

sequence of elements of the type “*fmiScalarVariable*” as shown in Figure 5.

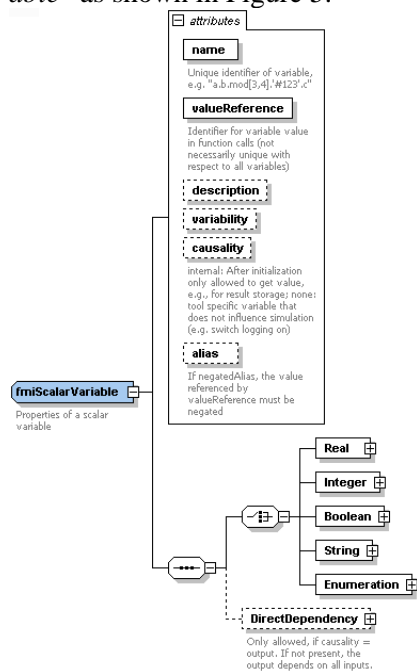


Figure 5: Structure of the *fmiScalarVariable* element

The information whether a variable is an input or output to the component (and therefore interface relevant) is coded in the optional attribute “*causality*” (condition: “*causality = input/output*”). Parameters for potential calibration of the component are identified through the “*variability*” attribute (condition: “*variability = parameter*”). FMI inputs/outputs map to the AUTOSAR sender-receiver port interface (input maps to “*RPort*” and output maps to “*PPort*”).

AUTOSAR supports different flavors of sender-receiver port communication (explicit/implicit communication, queued or un-queued communication, sending/receiving of data or events). There is no counterpart for these options in the FMI standard. Consequently, the desired mapping needs to be decided at the FMU import. For the further discussion we assume that FMI inputs/outputs are mapped to explicit, un-queued, data communication ports.

4.3 Mapping of FMI parameters to AUTOSAR calibration ports

In FMI a parameter is identified by the condition “*variability = parameter*” within the variable definition (see above). Parameters can be set before initializing the FMU. After initialization they are fixed and may not change during runtime.

In embedded automotive software design, manipulation of parameters is termed calibration. AUTOSAR provides flexible support for manipulating calibration parameters.

- Port-based calibration: Parameters are explicitly visible on the VFB. This mechanism is meant for public parameters of a SW-C (e.g. in Figure 8 the parameters for the PI-controller are port-based, public parameters).
- Private calibration parameters: These reside internally within a SW-C. They are not explicitly visible on the VFB level.

The rationale for differentiating between “private” and “public” parameters is that a supplier might want to indicate which parameters are safe to be calibrated by the OEM and which parameters the OEM should better not touch. Additionally, AUTOSAR allows to specify whether parameters may be calibrated online (while the software function is running), or only before initialization.

Like the previous mapping of FMI inputs/outputs to AUTOSAR ports, there is no univocal mapping from FMI parameters to AUTOSAR calibration parameters. However, it seems to be reasonable to map FMI parameters to “public” calibration ports, explicitly visible at VFB level³.

4.4 Wrapping the FMU C-code into an AUTOSAR Runnable Entity

Through its ports, the AUTOSAR SW-C specifies which information it requires from and provides to other components. The actual implementation of a component consists of a set of “runnable entities” (in short runnable⁴), which are code sequences in the SW-Cs that are activated through events, like timers or the receiving of data.

In order to execute an FMU as an AUTOSAR SW-C, it is necessary to wrap the C-function calls to the FMU into an AUTOSAR runnable.

Every runnable entity provides an entry point and an associated set of data. For components implemented using C or C++ the entry point of a runnable is implemented by a function with global scope defined in the source code of the software component. The RTE is the sole entity that can trigger the execu-

³ The current limitation of the FMI standard to allow parameters only to be set before initialization is in contrast to the well-established practice of online-calibration of controller algorithm parameters. Hopefully, future versions of the FMI standard will deal with that limitation.

⁴ A runnable runs in the context of a task. The task provides the common resources to the runnables such as context and stack-space. On the operating system level a task can be realized as either a full process or as a light-weight thread.

tion of a runnable. In [13], p.141 the signature of this function is defined as

```
<void|Std_ReturnType> <name>((IN RTE_Instance <instance>),
                             [role parameters])
```

AUTOSAR provides various events that can trigger a runnable (e.g. TimingEvent, DataReceivedEvent, DataReceiveErrorEvent, DataSendCompletedEvent, etc.). For using Modelica/FMU controller models in AUTOSAR applications the cyclic invocation plays the most important role. For that purpose the TimingEvent is used as activation method for FMU models.

Since the AUTOSAR activation of runnables is targeted at discrete controllers it does not support the concept of a solver, which is of course needed in the FMI specification. As a consequence, an adequate FMI solver must be wrapped inside the runnable functions. A design decision is needed whether FMUs with continuous states (“*numberOfContinuousStates* > 0”) shall be supported by the AUTOSAR importer, or if the import is restricted to purely discrete FMUs (superseding the need of wrapping a numerical integrator into the runnable). For the purpose of this work it is decided to only allow purely discrete FMUs⁵.

Notably, FMI 1.0 does not include an attribute for specifying a fixed sample period⁶. Thus, the sample period for the TimingEvent needs to be given as a parameter within the FMU import process.

5 Example application

The FMI to AUTOSAR conversion will be demonstrated in an application example. In this scenario we will consider export of a Modelica controller from Dymola through FMI. The exported FMU will then be converted to AUTOSAR and imported into the AUTOSAR Builder tool.

In order to focus the discussion, a simple, instructive example of a controlled drive is used. The example is modeled in Modelica using the Dymola tool (see Figure 6). The reference trajectory is provided

⁵ This restriction is not as severe as it may seem on first sight. If it is desired to use models with continuous states, some tools (e.g. Dymola) provide options of exporting such models as FMUs with inline integrators. As a result the exported FMU has no external continuous states (“*numberOfContinuousStates* = 0”), thus no integrator needs to be provided for executing such an FMU.

⁶ Hopefully, future versions of the standard will allow specifying a fixed sample period.

by the “reference” block. The “*piController*” block implements the closed-loop control of the plant.

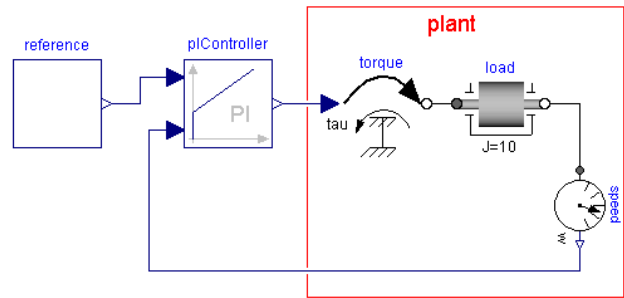


Figure 6: Simple controlled drive example as Modelica model in Dymola

The PI-controller (proportional-integral controller) may be parameterized with the proportional gain “*k*” and the time constant “*T*” of the integral term, as shown in Figure 7.

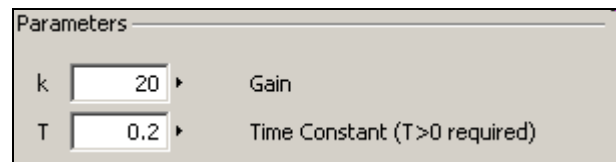


Figure 7: Parameter dialog for controller calibration in Dymola

Figure 8 shows how the example can be modeled within an AUTOSAR VFB diagram. The parameters are modeled as explicit inputs to the “*PIController*” SW-C. The “*PI_Init*” runnable initializes the controller and sets the provided parameters. The “*PI_Run*” runnable is called periodically to provide the required actuating variable. Instead of the plant, Sensor-Actuator SW-Cs have been introduced (“*TorqueActuator*” and “*SpeedSensor*”).

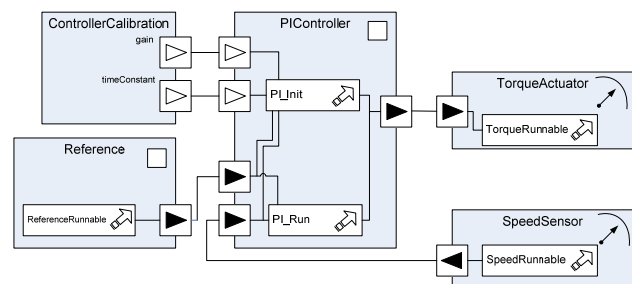


Figure 8: Simple controlled drive example as AUTOSAR VFB diagram (including sensor and actuator components, as well as parameter ports for controller calibration)

The Modelica PI-controller is exported from Dymola as an FMU and transformed from the FMI schema to the AUTOSAR schema. Similarly, the required C wrapper code for the AUTOSAR runnable is automatically generated from the FMI schema. In both cases Scala and Java are used as the implementation languages of choice for carrying out

these transformations⁷. In Figure 9 an excerpt of the Model Description File of the PI-controller as exported by Dymola is given.

```
<?xml version="1.0" encoding="UTF-8"?>
<fmiModelDescription
  fmiVersion="1.0"
  modelName="PI"
  modelIdentifier="PI"
  guid="{8362d077-2c00-4f74-975b-339843f12ce7}"
  variableNamingConvention="structured"
  numberOfContinuousStates="0"
  numberOfEventIndicators="0">
  <ModelVariables>
    <ScalarVariable
      name="k"
      valueReference="16777216"
      description="Gain"
      variability="parameter">
      <Real start="20"
        fixed="true"/>
    </ScalarVariable>
    <ScalarVariable[]>
    <ScalarVariable[]>
    <ScalarVariable
      name="u_ref"
      valueReference="352321536"
      causality="input">
      <Real/>
    </ScalarVariable>
    <ScalarVariable[]>
    <ScalarVariable
      name="y"
      valueReference="335544320"
      causality="output">
      <Real/>
    </ScalarVariable>
    <ScalarVariable[]>
    <ScalarVariable[]>
  </ModelVariables>
</fmiModelDescription>
```

Figure 9: Excerpt from the Model Description File of the PI-controller (FMI schema compliant xml format)

The necessary workflow for transforming the FMU to an AUTOSAR SW-C is depicted in Figure 10. The workflow is highly automated, since the current version of the fmi2autosar program needs no user interaction except of specifying the location of the program’s input and the desired fixed sample period. Basically, the import into AUTOSAR Builder works by just copying the files generated by fmi2autosar into an AUTOSAR project directory and “refreshing” the project⁸. The screenshot in Figure 11 shows the AUTOSAR Master Editor view, after the PI-controller import.

⁷ The implementation effort was considerably reduced by leveraging the functionality of the XMLSpy tool [11] to automatically generate XML data bindings for the Java language.

⁸ For further processing in AUTOSAR Builder, e.g., simulation on VFB level and RTE generation, necessary build dependencies and compiler flags need to be configured manually in AUTOSAR Builder. Because the required settings are highly tool- and application-specific no attempt is made to provide default settings.

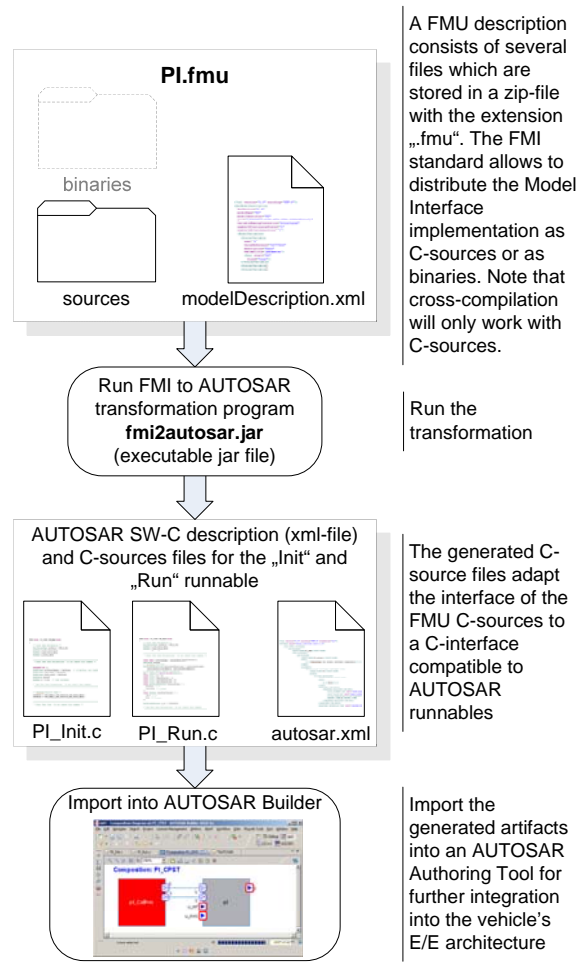


Figure 10: FMU to AUTOSAR-SW-C transformation workflow demonstrated through the PI-controller example

AUTOSAR allows a flexible structuring of elements through the use of packages and subpackages. To achieve a well-arranged layout, which facilitates integration into an AUTOSAR project, the proposed transformation collects all elements resulting from an FMU transformation into one package. The value of the FMU’s “modelIdentifier” attribute is used as base string for the package and subpackage names (see Figure 11).

After the import the model can be further processed in AUTOSAR Builder. It can be integrated with other SW-Cs and simulated on the VFB level using the Geensoft ASim tool.

6 Conclusions

This paper has presented a mapping and conversion scheme between the Functional Mockup Interface (FMI) for model exchange and the automotive software architecture standard, AUTOSAR. A suitable subset of the AUTOSAR software component speci-

fication was selected for the mapping and the rationale for these decisions was motivated. The design has been validated by importing the transformed FMI models into an AUTOSAR Authoring Tool and simulating the design on the Virtual Functional Bus level.

The FMI to AUTOSAR mapping process has also identified missing features in FMI that should be worth considering for future versions of the standard.

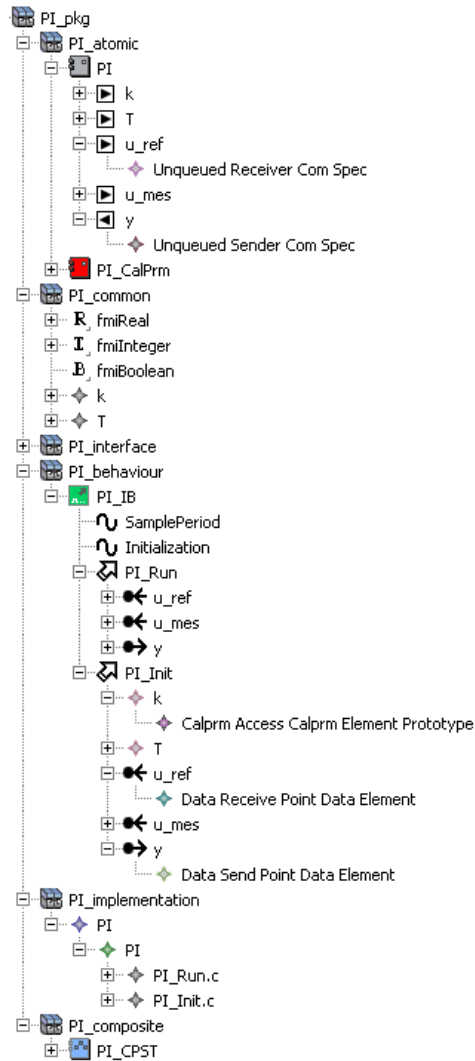


Figure 11: Screenshot of the AUTOSAR Master Editor after importing the FMU of the PI controller into the Geensoft AUTOSAR Builder tool

6.1 Acknowledgements

Partial financial support of DLR by BMBF (BMBF Förderkennzeichen: 01IS08002) for this work within the ITEA2 project MODELISAR is highly appreciated.

Dassault Systèmes AB thanks the Swedish funding agency VINNOVA for partial funding of this

work within the ITEA2 project MODELISAR (2008-02291).

References

- [1] Dymola Version 7.4. Dassault Systèmes, Lund, Sweden (Dynasim). Homepage: <http://www.dymola.com>, 2010
- [2] AUTOSAR Builder Version 2010-2a. Dassault Systèmes, Brest, France (Geensoft). Homepage: <http://www.geensoft.com/>, 2010
- [3] Jörg Schäuffele, Thomas Zurawka. Automotive Software Engineering, SAE International, 2005
- [4] Hilding Elmqvist, Martin Otter, Dan Henriksson, Bernhard Thiele, Sven Erik Mattsson. Modelica for Embedded Systems, 7th Int. Modelica Conference, 2009
- [5] Barbara Lange. Verbunden; Austauschformat für die Simulation, iX extra 10/2010, p. VIII
- [6] MODELISAR consortium. Functional Mock-up Interface for Model Exchange 1.0, http://modelisar.org/specifications/FMI_for_ModelExchange_v1.0.pdf, 2010
- [7] AUTOSAR GbR. Technical Overview, AUTOSAR, Part of Release 3.1, AUTOSAR_TechnicalOverview.pdf, 2008
- [8] W3C. XML Schema Part 1: Structures Second Edition. W3C Recommendation 28 October 2004, <http://www.w3.org/TR/xmlschema-1/>
- [9] MapForce 2010, Altova GmbH, Vienna, Austria, <http://www.altova.com/mapforce.html>, 2010
- [10] Martin Odersky. The Scala Language Specification Version 2.8, <http://www.scala-lang.org/docu/files/ScalaReference.pdf>, 9 November 2010
- [11] XMLSpy 2010, Altova GmbH, Vienna, Austria, <http://www.altova.com/xmlspy.html>, 2010
- [12] AUTOSAR GbR. Software Component Template, Part of Release 3.1, AUTOSAR_SoftwareComponentTemplate.pdf, 2010
- [13] AUTOSAR GbR. Specification of RTE, Part of Release 3.1, AUTOSAR_SRS_RTE.pdf, 27.01.2010

Combining Advantages of Specialized Simulation Tools and Modelica Models using Functional Mock-up Interface (FMI)

Yongqi Sun, Stephanie Vogel, Haiko Steuer

Siemens AG, Energy Sector, Erlangen, Germany

yongqi.sun@siemens.com, vogel.stephanie@siemens.com, haiko.steuer@siemens.com

Abstract

In power plant applications, detailed transient analysis of the evaporator results in very large fluid systems, i.e. the model consists of many equations. This is because the evaporator has many tubes and the spatial discretization has to be quite fine in order to appropriately model transient evaporation processes. In such cases, due to performance and workflow reasons, we use a specialized in-house tool Dynaplant [1]. Coupling between Dynaplant and a Modelica-simulator helps to benefit in addition from the possibility of rapid development of new components in Modelica.

Here, it is shown, how the Functional Mock-up Unit (FMU) export from a Modelica model can be used to perform a co-simulation with Dynaplant and an FMU simulator. The FMU is defined via the Functional Mock-up Interface (FMI) for Model Exchange v1.0 [2]. Advantages but also restrictions and challenges are covered.

Keywords: Fluid; Co-simulation; FMU; FMI

1 Introduction

1.1 Modelica world versus Dynaplant

Modelica is the preferred modeling language for dynamic simulations within Siemens Energy [3] due to the high degree of maintainability of Modelica models. In addition, new models can rapidly be developed.

Dynaplant is highly specialized for large fluid systems (e.g. detailed evaporator models). Its performance and usability fulfills our requirements: Performance is increased by using fast water/steam property functions and taking advantage of multi-core CPUs. Usability features are data interface to steady state design tool and restart ability. However, the model library is quite restricted and the development of new Dynaplant model is time-consuming.

1.2 Use cases, where best of two worlds is needed

(A) Typically, there is already an evaporator model in Dynaplant, such that the evaporator itself can easily be simulated in Dynaplant. Now assume some urgent analysis request, which could be answered via a transient simulation of the evaporator together with some neighboring components. There is not yet any Dynaplant model of this neighboring component, but it can be rapidly developed in Modelica.

In this case, some coupling between Dynaplant and Modelica models will be helpful. Here, the process interface between Modelica and Dynaplant may consist of a mass flow rate m_flow and enthalpy h from the Dynaplant evaporator and a pressure p from the new Modelica component.

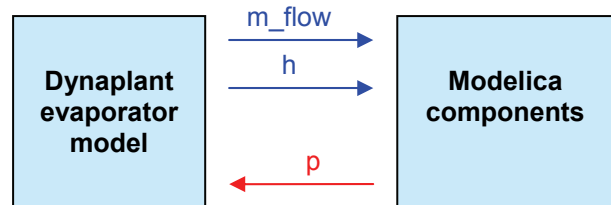


Figure 1: Typical use case (A) for coupling Dynaplant with Modelica models.

Formerly, in such a case an iterative procedure had been carried out. Here, the Dynaplant and the Modelica model are solved alternating with time tables containing the output of one model as input for the other. After several iterations hopefully a convergence is observed. This iterative procedure is very time-consuming and does not guarantee any convergence at all.

(B) The power plant block control is implemented in another in-house software SPPA-T3000 [4]. Currently, there is a T3000-Modelica parser under development generating Modelica models out of SPPA-T3000. This way, the block control could be easily modeled in Modelica. Then, the steam generator (and may be steam turbines) would be modeled in

Dynaplant and remaining parts of the power plant including block control would be modeled in Modelica. This enables the solution of the following use cases for coupling Dynaplant and Modelica models:

- Testing of SPPA-T3000 control schemes using detailed Dynaplant process models
- Dynaplant process simulation using original SPPA-T3000 control schemes

2 Routes to couple in-house tools with Modelica using FMU

In the following two sections 2.1 and 2.2, the Functional Mock-up Unit and its in-house simulator will be introduced. Sections 2.3 and 2.4 present possibilities to couple Modelica and Dynaplant models.

2.1 Introduction to FMU

The FMU is an implementation of the FMI for Model Exchange [2] definitions. We used this instead of the alternative specification (FMI for Co-Simulation), since our long-term goal is to completely include the FMU model into a single simulation environment. The intention is that dynamic system models of different software systems (including Modelica and non-Modelica tools) can generate C-Code of a dynamic system model, which can be used for simulation.

An FMU-file is a zipped archive which contains at least:

1. A small set of easy to use C-functions for all the needed model equations such as differential, algebraic and discrete equations with events. These C-functions can either be provided in source and/or binary form.
2. An xml-file containing the definition of all variables in the model and other model information needed to simulate the model.

The main features of FMU are:

1. It is possible to utilize several instances of a model and to connect models hierarchically together.
2. A model is independent of the target simulator because it does not use a simulator specific header file as in other approaches.
3. An FMU may either be self-integrating or require an external solver to perform numerical integration.

Assume an FMU without self-integration. For coupling this FMU with another simulator (e.g.

Dynaplant), one may either connect an FMU simulator with the other simulator (co-simulation) or add the FMU equations directly into the equation system of the other simulator (single model, single solver). The FMU simulator needed for co-simulation is described in the next section. The remaining two sections cover both approaches mentioned.

2.2 Sadida: An Interface to FMU

Sadida is our in-house FMU simulator. It contains solvers for integrating a given FMU. The model equations from the FMU and the time integration solver are wrapped together into several methods of a simulator class. The most important methods are:

- Setup: Allocate memory for the FMU model, and initialize the integrator.
- SetVariable
- GetVariable
- InitializeModel: Computes the initial state corresponding to the FMU initial equations.
- ReAllocateModel
- Run: Perform time integration with required time span and time step size.

Using Dymola 7.4 [5] with code export option, it is possible to export a FMU from a Modelica model. Our in-house FMU simulator Sadida then enables us to integrate any Modelica model. The methods of this FMU simulator can be called by Dynaplant.

2.3 Co-simulation: FMU simulator included in Dynaplant

The first approach uses co-simulation in order to couple the FMU model with the Dynaplant model. Usually, a co-simulation is controlled by a tool like TISC [6] organizing data exchange between several simulators. In contrast, here, another kind of co-simulation is done where Dynaplant (one of the simulation partners) takes control of the co-simulation:

Dynaplant calls methods from the FMU simulator in order to manage data exchange and provides time step sizes for both simulation partners.

The co-simulation is prepared using the following procedure:

- Create an instance of the Sadida FMU simulator class. The following steps are done using methods from this FMU simulator.
- Import FMU model
- Modify FMU model parameters
- Set input variables $u(t)$ of FMU model from Dynaplant's initial state

- Initialize the FMU model
- Get initial values of output variables $y(0)$ from the FMU model

A Dynaplant integration of the time interval $t_n \dots t_{n+1}$ is carried out using constant output $y(t_n)$ from Sadida. After each such Dynaplant time step, the following FMU simulator methods are called by Dynaplant via Sadida:

- Set input variables $u(t_{n+1})$ from Dynaplant.
- Integrate FMU model over time interval $t_n \dots t_{n+1}$ using constant $u(t_{n+1})$.
- Get output variables $y(t_{n+1})$ from FMU model

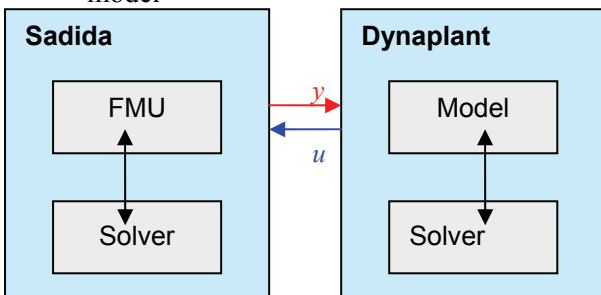


Figure 2: Co-simulation: The FMU model is solved by the FMU simulator Sadida, the Dynaplant model is solved by Dynaplant. Between both solvers, data exchange takes place.

This way, a co-simulation of both Sadida and Dynaplant model can be performed. The interaction of models and solvers is shown in Figure 2. The timing is illustrated in Figure 3. It leads to time discretization errors.

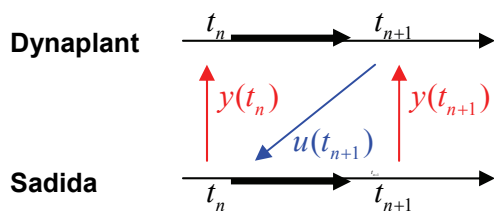


Figure 3: Timing of both Dynaplant and Sadida solvers. First Dynaplant integrates a time step $t_n \dots t_{n+1}$ assuming constant $y(t_n)$ from Sadida. Then, Sadida integrates this time interval using constant $u(t_{n+1})$ from Dynaplant. This results in time discretization errors.

2.4 Next step: FMU equations included in Dynaplant (single solver)

As shown in Figure 4, the next step will go beyond co-simulation. Rather than solving the FMU model

with a second simulator, the FMU equations will directly be added to the Dynaplant equation system and solved with the Dynaplant solver. This way, there will be no time discretization errors any more.

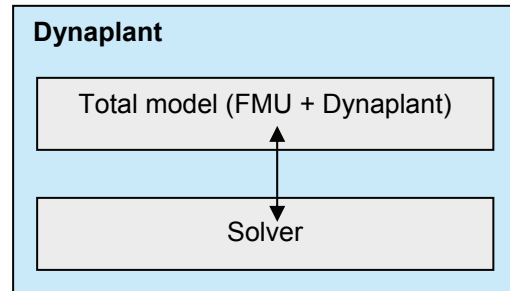


Figure 4: Next step: FMU equations are added to the Dynaplant model and the total model is solved by a single solver.

This approach is just an outlook with many open questions. Especially the following Dynaplant features have to be clarified:

- Fixed Jacobian structure (using potential Jacobian non-zeros)
- Restart ability
- Nominal values dependent on type of variable (e.g. fluid pressure)
- Less sophisticated event handling

3 Co-simulation Restrictions and Challenges

So far the co-simulation approach for coupling Dynaplant and Modelica models is realized. In this chapter we share our experiences including restrictions and challenges.

3.1 Restrictions of co-simulation based on FMU

Due to the time discretization errors caused by the exchanged variables, time intervals $t_n \dots t_{n+1}$ should not be too large. Dynaplant uses time step adjustments. The resulting time steps have to be restricted to an appropriate upper limit: $t_{n+1} - t_n \leq \Delta t_{max}$.

The following restriction is Dymola specific but motivates an extension of the FMI: An FMU exported by Dymola 7.4 without “code export option” will result in an initialization failure on a PC without Dymola. If there are license restrictions in the FMU, it would be better to include license check methods to the FMI rather than computational methods returning FALSE.

A rule deserving notice is that, after the model has been initialized, no parameter or constant vari-

able of the model allows its value to be changed. `ReAllocateModel` must be called if there is need changing parameter. The values of the other variables can be changed at any time step.

3.2 Challenges

In this section some problems with the co-simulation approach are mentioned.

Consistent initialization: In order to obtain a consistent initial state of the FMU model, the inputs $u(0)$ from Dynaplant's initial state have to be set first (by the way: there was a similar bug in Dymola 7.4 for which a patch is available). A consistent initialization of both FMU and Dynaplant model in total is not yet realized, since the output of the FMU initialization may influence Dynaplant's initial state.

Restart ability: One of the advantages of Dynaplant is its ability to load the final state of a former simulation as initial state of another simulation. Until now, the final state of the FMU is not saved for this purpose, such that the initial equations of the FMU may be adapted for a restart.

Support of more than one FMU: Until now only one FMU is supported. It will be possible to enable several FMUs by creating a simulator instance for each single FMU. However, since there have not yet been a use case for this feature, it is not yet resolved.

Fluid connector interface: At the co-simulation interface, there is no fluid connector like in the Modelica Standard Library. Instead real inputs and outputs are used, assuming no flow reversal.

Accessibility of FMU variables for output: Until now, not all variables of the FMU can be seen in the Dynaplant output. FMU variables which shall enter the Dynaplant output have to be selected individually. A tree view of all FMU variables would be more usable.

3.3 First experiences

With the help of the co-simulation between FMU and in-house simulator, a clear improvement of the productivity was reached in comparison to the iterative procedure (by factor 5). Up to now the co-simulation seems to be robust, indeed, an easy Modelica model (with few states) was used that individually simulated also has no convergence problems.

A disadvantage is absolutely that the FMU is like a black box and it is very difficult to identify errors in the FMU model. Usability will be enhanced when all FMU variables are accessible in the Dynaplant output.

4 Conclusions

Coupling of Modelica models with in-house tools combines advantages of both specialized simulation tools and flexible Modelica models. The FMU export of the Modelica model is helpful, since it contains standard interfaces to exchange information with the model.

In our co-simulation approach, an FMU simulator is used to integrate the FMU model. Dynaplant takes control over the time step adjustments and uses `set`, `run` and `get` methods to exchange variables with the FMU simulator. This approach is realized and reveals promising advantages.

Next step will go beyond co-simulation: It will be even better to use the FMU gathering its equations in order to add them to the equation system of the specialized simulation tool. In contrast to co-simulation, the total model will be integrated by a single solver.

Tools importing FMUs should be enabled to generate meaningful error messages in case of license failures. Therefore we propose to add some license check method to the FMI.

Support by German Ministry BMBF (BMBF Förderkennzeichen: 01IS09029C) within the ITEA project OPENPROD [7] is gratefully acknowledged.

References

- [1] K. Link, H. Steuer, A. Butterlin, [Deficiencies of Modelica and its simulation environments for large fluid systems](#), Proceedings 7th International Modelica Conference, Como, Italy, Sep 20-22, 2009
- [2] Functional Mock-up Interface for Model Exchange [FMI for ModelExchange_v1.0.pdf](#) (Jan 26, 2010)
- [3] www.energy.siemens.com
- [4] www.energy.siemens.com/hq/en/automation/power-generation/sppa-t3000.htm
- [5] [Dymola 7.4](#)
- [6] TISC from www.tlk-thermo.com
- [7] www.openprod.org

Symbolically Derived Jacobians Using Automatic Differentiation - Enhancement of the OpenModelica Compiler

Willi Braun Lennart Ochel Bernhard Bachmann

Bielefeld University of Applied Sciences, Department of engineering and mathematics

Am Stadtholz 24, 33609 Bielefeld

{wbraun,lochel,bernhard.bachmann}@fh-bielefeld.de

Abstract

Jacobian matrices are used in a wide range of applications - from solving the original DAEs to sensitivity analysis. Using Automatic Differentiation the necessary partial derivatives can be provided efficiently within a Modelica-Tool. This paper describes the corresponding implementation work within the OpenModelica Compiler (OMC) to create a symbolic derivative module. This new OMC-feature generates symbolically partial derivatives in order to calculate Jacobian matrices with respect to different variables. Applications presented here, are the generation of linear models of non-linear Modelica models and the usage of the Jacobian matrix in DASSL for simulating a model.

Keywords: Symbolic Jacobian, Automatic Differentiation, Linearization, DASSL, OpenModelica

1 Introduction

In the process of modeling and simulation the usage of derivatives in many stages of this process is very common. The derivatives are useful for simulating a model as well as for the sensitivity analysis [1] or the optimization [4] of models. The derivatives can be calculated in different ways. There exist numerical methods like finite difference, or symbolical methods as in algebra systems. But there is another method containing characteristics of both of them: Automatic Differentiation (AD) is the better choice over other ways for computing derivatives. It is accurate like symbolic differentiation, since the results are not affected by any truncation errors. AD is originally a numerical method

in contrast to numerical differentiation that evaluates the derivative of a function specified by sequence of assignments in a computer program. Since a Modelica program is written with symbolic expressions, AD can be used to calculate symbolically partial derivatives. In this work the OMC is enhanced to provide the symbolic derivatives for a Modelica model using AD. The new OMC feature is applicable in a versatile way. As first application it is used for the linearization of non-linear models. The linearization of a non-linear model needs the calculation of partial derivatives with respect to some specific variables of the Modelica model. The partial derivatives are organized in so-called Jacobian matrices. For the linear model it must be calculated four different Jacobian matrices so that the main task is the calculation of symbolic partial derivatives for the linearization. A further application of this new OMC capability, is the usage of the derivatives for simulating a Modelica model. The commonly used implicit integration method DASSL is providing an interface for the symbolic Jacobian matrix. This feature can be now used to speed-up the solving time in OMC.

The structure of this paper is as follows: First, methods from AD theory are shortly introduced in order to calculate the symbolic derivatives. Afterwards a short introduction of the relevant Modelica language features is presented and the mathematical representation of the corresponding Modelica models is described. With this implementation we are able to differentiate almost the complete Modelica language elements supported by OpenModelica. Finally, the generation of a linear model and the usage of the symbolic derivatives for simulating a Modelica model with DASSL are presented as applications.

2 Automatic Differentiation

The calculation of the symbolic derivatives of a Modelica model is possible by using automatic differentiation (AD) methods. AD is an efficient method to calculate the derivative value for an algorithmic function. This technique is based on the fact that the derivatives of a function can be calculated by repeatedly applying mathematical rules to all the sequential elementary operations of a coded function. The elementary operations can be differentiated by applying the basic derivation rules

$$\begin{aligned} \nabla(u \pm v) &= \nabla u \pm \nabla v \\ \nabla(uv) &= u\nabla v + v\nabla u \\ \nabla\left(\frac{u}{v}\right) &= \frac{(\nabla u - \frac{u}{v}\nabla v)}{v} \end{aligned}$$

for the arithmetic operations and the chain rule

$$\nabla\phi(u) = \dot{\phi}(u)\nabla u$$

for differentiable functions ϕ (e.g. such as the standard functions $\sin(x), \cos x, \dots$) with known derivatives. This approach is referred to in literature as "forward" mode [9].

For example, a function given by the formula 1 can be decomposed in the elementary operations as in table 1.

$$f(x_1, x_2) = (x_1 * x_2 + \sin(x_1))(3 * x_1^2 + x_2) \quad (1)$$

The basic rules of differentiation can be applied to the decomposed arithmetic operations to obtain the partial derivative of the function. Thus, the final results are the values $t_9 = f(x_1, x_2)$ of the function and its partial derivatives $\nabla f = \nabla t_9 = [(t_2 + \cos(t_1))t_8 + 6t_1t_5, t_1t_8 + t_5]$.

Since AD is originally a numerical method, it is common to determine the values only. If these terms are replaced by the original expressions that are available inside a Modelica compiler the symbolic derivative formulas are obtained. This automatic differentiation method can be used analogically in order to calculate the partial derivatives to the optimized DAEs as they occur in Modelica. This is possible, because the calculation of partial derivatives is performed by consistently applying the chain rule and the basic differentiation rules as mentioned above.

3 Differentiate a Modelica Model

A Modelica model is typically translated to a basic mathematical representation in terms of a flat system of differential and algebraic equations before being able to simulate the model. This translation process elaborates on the internal model representation by performing analysis and type checking, inheritance and expansion of base classes, modifications and redeclarations, conversion of connect equations to basic equations, etc. The result of this analysis and translation process is a flat set of equations, including conditional equations as well as constants, variables, and function definitions. By the term flat is meant that the object-oriented structure has been broken down to a flat representation where no trace of the object hierarchy remains, apart from dot notation (e.g. `Class.Subclass.variable`) within names.

Flat Modelica DAEs could be represented mathematically by the equation:

$$0 = F(\dot{\underline{x}}(t), \underline{x}(t), \underline{u}(t), \underline{y}(t), \underline{p}, t) \quad (2)$$

Below the notations used in the equation above are summarized:

- $\dot{\underline{x}}(t)$ the differentiated vector of state variables of the model.
- $\underline{x}(t)$ the vector of state variables of the model, i.e., variables of type Real that also appear differentiated somewhere in the model.
- $\underline{u}(t)$ a vector of input variables, i.e., not dependent on other variables, of type Real. They also belong to the set of algebraic variables since they do not appear differentiated.
- $\underline{y}(t)$ a vector of Modelica variables of type Real which do not fall into any other category.
- \underline{p} a vector containing the Modelica variables declared as parameter or constant i.e., variables without any time dependency.

This implicit equation is transformed to the explicit state-space representation by the so-called block-lower-triangular (BLT) transformation resulting in the optimized DAEs. This transformation is done by a matching and sorting algorithm which results in a sequence of assignments so that the variables can be solved sequentially [7]:

Operations	Differentiate($t_i, \{x_1, x_2\}$)	∇f
$t_1 = x_1$	$\nabla t_1 = [1, 0]$	$[1, 0]$
$t_2 = x_2$	$\nabla t_2 = [0, 1]$	$[0, 1]$
$t_3 = t_1 t_2$	$\nabla t_3 = t_1 \nabla t_2 + \nabla t_1 t_2$	$[t_2, t_1]$
$t_4 = \sin(t_1)$	$\nabla t_4 = \cos(t_1) \nabla t_1$	$[\cos(t_1), 0]$
$t_5 = t_3 + t_4$	$\nabla t_5 = \nabla t_3 + \nabla t_4$	$[t_2 + \cos(t_1), t_1]$
$t_6 = t_1 * t_1$	$\nabla t_6 = 2t_1 \nabla t_1$	$[2 * t_1, 0]$
$t_7 = 3 * t_6$	$\nabla t_7 = 3 \nabla t_6$	$[6 * t_1, 0]$
$t_8 = t_7 + t_2$	$\nabla t_8 = \nabla t_7 + \nabla t_2$	$[2 * t_1, 1]$
$t_9 = t_5 * t_8$	$\nabla t_9 = t_5 \nabla t_8 + \nabla t_5 t_8$	$[t_2 + \cos(t_1) t_8 + 6 t_1 t_5, t_1 t_8 + t_5]$

 Table 1: Decomposed function $f(x_1, x_2)$ to elementary operations and the partial derivatives.

$$\begin{aligned}
 0 &= F(\dot{\underline{x}}(t), \underline{x}(t), \underline{u}(t), \underline{y}(t), \underline{p}, t) \\
 0 &= F(\underline{z}(t), \underline{x}(t), \underline{u}(t), \underline{p}, t), \quad \underline{z}(t) = \begin{pmatrix} \dot{\underline{x}}(t) \\ \underline{y}(t) \end{pmatrix} \\
 \underline{z}(t) &= \begin{pmatrix} \dot{\underline{x}}(t) \\ \underline{y}(t) \end{pmatrix} = g(\underline{x}(t), \underline{u}(t), \underline{p}, t) \\
 \begin{pmatrix} \dot{\underline{x}}(t) \\ \underline{y}(t) \end{pmatrix} &= \begin{pmatrix} h(\underline{x}(t), \underline{u}(t), \underline{p}, t) \\ k(\underline{x}(t), \underline{u}(t), \underline{p}, t) \end{pmatrix} \quad (3)
 \end{aligned}$$

This sequence of assignments can immediately be used for calculating the partial derivatives symbolically by means of automatic differentiation. Thus the differentiation process is performed on such optimized DAEs. These DAEs are separated in two partitions, a state block and an algebraic block. The function h represents the state block and consists of all equations, which are necessary to determine the differentiated states. The function k represents the algebraic block, which contains all remaining equations.

Consider, for example, the following small differential-algebraic system:

$$\begin{aligned}
 f_1 &:= \dot{x}_1 = a * x_1 \\
 f_2 &:= \dot{x}_2 = a * x_2 + \dot{x}_1 \\
 f_3 &:= a = \sin(x_1) + \cos(x_2)
 \end{aligned}$$

To calculate all necessary partial derivatives the system has to be sorted by the BLT-Transformation based on the adjacency matrix:

$$\begin{array}{ccc}
 \dot{x}_1 & \dot{x}_2 & a \\
 f_1 \begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} & & f_3 \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix}
 \end{array}$$

To get all partial derivatives with respect to the states the whole system needs to be differentiated, which means every equation has to be differentiated with respect to the states. This requires the derivatives of all known variables with respect to the states. In a Modelica model it is assumed that the known variables are the states and the inputs. In this example only the states appear as known variables:

$$\begin{pmatrix} \frac{\partial x_1}{\partial x_1} = 1 & \frac{\partial x_2}{\partial x_1} = 0 \\ \frac{\partial x_1}{\partial x_2} = 0 & \frac{\partial x_2}{\partial x_2} = 1 \end{pmatrix}$$

The next step is to take the sorted equations and differentiate straight forward applying the rules described above. With all the resulting partial derivatives it is possible to organize the Jacobian matrix with respect to the states \underline{x} in equation (4). In the following, Modelica language features are described which need to be handled by the automatic differentiation.

Equations

The differentiation of ordinary equations is straightforward. In the optimized DAEs the matching algorithm provides information about the variable which has to be solved for in each equation. Therefore, the equations are rearranged to a corresponding assignment and differentiated. This also works for equations including if-expressions, where each branch will be differentiated, respectively. Non-linear equations will result into equations depending linearly on the differentiated variables (see example in the Algebraic loop section).

$$A = \frac{\partial f}{\partial \underline{x}} = \begin{pmatrix} \frac{\partial f_1}{\partial x_2} & \frac{\partial f_1}{\partial x_2} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} \end{pmatrix} = \begin{pmatrix} \cos(x_1)x_1 + a & -\sin(x_2)x_1 \\ \cos(x_1)x_2 + \cos(x_1)x_1 + a & -\sin(x_2)x_2 + a - \sin(x_2)x_1 \end{pmatrix} \quad (4)$$

Algebraic loops

In many applications the transformation to the optimized DAEs cannot achieve a true lower-triangular form. It is at least possible to reduce the DAEs to a Block-Lower-triangular form with diagonal blocks of minimal size. These blocks are called algebraic loops and must be solved simultaneously. In general, this results in a system of linear and/or nonlinear equations.

For example the following equations have to be solved simultaneously:

$$\underline{f}(\underline{x}, \underline{p}, t) := \begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \end{pmatrix} = \begin{pmatrix} ax_1 + \frac{1}{2}\dot{x}_2^2 \\ bx_2 - \frac{1}{2}\dot{x}_1^2 \end{pmatrix}$$

The equations are differentiated with respect to the state to determine the first row of the Jacobian matrix:

$$\begin{pmatrix} \frac{\partial f_1}{\partial x_1} := \frac{\partial \dot{x}_1}{\partial x_1} = a + \frac{\partial \dot{x}_2}{\partial x_1} \dot{x}_2 \\ \frac{\partial f_2}{\partial x_1} := \frac{\partial \dot{x}_2}{\partial x_1} = -\frac{\partial \dot{x}_1}{\partial x_1} \dot{x}_1 \end{pmatrix}$$

The resulting equation must still be solved simultaneously to determine the expressions for the first row of the Jacobian matrix. However, nonlinear equations that are differentiated, result always in equations depending linearly on the differentiated variables, which in this case, yield a linear system of equations to be solved.

Algorithms

Whereas equations are well suited to describe physical processes, there are situations where computations are more conveniently expressed by algorithms in a sequence of statements. In contrast to equations, statements are fixed assignments, i.e. the right-hand-side ones are assigned to the left-hand-side ones. Several assignments to the same variable can be performed in one algorithm section. Besides of simple assignment statements, an algorithm can contain if-, while-, and for-clauses. The symbolic differentiation can handle all of them.

Functions in Modelica

In Modelica, there exist two different types of functions, a Modelica function, written in Modelica

code, and external functions that are written in C/Fortran code. A Modelica function is defined by an algorithm section that can be differentiated in the same way as algorithms. From the result a new Modelica function as the derivative to the original one is generated. This derivative function can be propagated by the derivative annotation to other process that needs the derivative. For external functions the numerical finite difference method is used, if that functions do not provide partial derivatives with the aid of the derivative annotation.

4 Applications for Symbolic Jacobian

4.1 Linear Models

A general nonlinear Modelica model is represented by state-space equations with n state variables, m input variables and k output variables:

$$\begin{pmatrix} \dot{\underline{x}}(t) \\ \underline{y}(t) \end{pmatrix} = \begin{pmatrix} h(\underline{x}(t), \underline{u}(t), \underline{p}, t) \\ k(\underline{x}(t), \underline{u}(t), \underline{p}, t) \end{pmatrix}$$

Linearizing the state-space equations the Taylor series expansion is applied and leads to a continuous-time linear dynamical system that has the form:

$$\begin{aligned} \dot{\underline{x}}(t) &= A(t) * \underline{x}(t) + B(t) * \underline{u}(t) \\ \underline{y}(t) &= C(t) * \underline{x}(t) + D(t) * \underline{u}(t) \end{aligned}$$

$$\begin{aligned} A(t) &= \frac{\partial h}{\partial \underline{x}} \in \mathbb{R}^{n \times n}, B(t) = \frac{\partial h}{\partial \underline{u}} \in \mathbb{R}^{n \times m} \\ C(t) &= \frac{\partial k}{\partial \underline{x}} \in \mathbb{R}^{k \times n}, D(t) = \frac{\partial k}{\partial \underline{u}} \in \mathbb{R}^{k \times m} \end{aligned}$$

The matrices $A(t)$, $B(t)$, $C(t)$, and $D(t)$ are the Jacobian matrices of the non-linear Modelica model. Thus the finding of linearization of a model is done by the calculation of the Jacobian matrices at a convenient time.

After all, the linear model can easily be generated when it's possible to differentiate a set of equations with respect to a set of variables. Therefore functions are implemented, that apply the method of forward automatic differentiation to given sets of equations, algorithms and variables. This function can deal with single equations, with

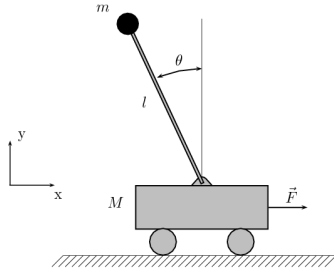


Figure 1: Schematic figure of inverse pendulum model.

systems of equations as well as algorithm sections and generate symbolically the needed Jacobian matrices for the linearization. The different steps of this procedure are sketched by the following model in Listing 1 and the corresponding schematic diagram of that model in Figure 1. In this model an inverse pendulum is balanced by a cart.

```

model InversePendulum
  parameter Real M = 0.5;
  parameter Real m = 0.2;
  parameter Real b = 0.1;
  parameter Real i = 0.006;
  parameter Real g = 9.8;
  parameter Real l = 0.3;
  parameter Real pi = 3.141592653589793;
  Real cart_x;
  Real cart_v;
  Real pendulum_theta;
  Real pendulum_w;
  output Real y[2];
  input Real u;
equation
  der(cart_x) = cart_v;
  der(pendulum_theta) = pendulum_w;
  (M + m)*der(cart_v) + b*cart_v +
  u = m*l*der(pendulum_w)*cos(pendulum_theta+pi)
  -m*l*pendulum_w^2*sin(pendulum_theta+pi);
  (i+m*l^2)*der(pendulum_w)+
  m*l*g*sin(pendulum_theta+pi)=
  -m*l*der(cart_v)*cos(pendulum_theta+pi);
  y={cart_x, pendulum_theta};
end InversePendulum;
    
```

Listing 1: InversePendulum model

The equations in Figure 2 are included into the generated simulation program and with this information the matrices A and C linearized model can be generated to any point in time. After compiling the generated C-code the evaluation of the linearized model at point in time 0 yields the Modelica model in Listing 2. The same simulation program can generate the linear model at any other point in time, i.e. equal 1, by simulating until then and afterwards evaluating the symbolic differenti-

ated equations at this point.

```

model linear_InversePendulum
  parameter Integer n = 4; // states
  parameter Integer k = 1; // top-level inputs
  parameter Integer l = 2; // top-level outputs
  parameter Real x0[4] = {0,0,0,0};
  parameter Real u0[1] = {0};
  parameter Real A[4,4] =
  [0,1,0,0;
  0,-0.1818181818181819,2.672727272727272,0;
  0,0,0,1;
  0,-0.4545454545454546,31.18181818181818,0];
  parameter Real B[4,1] =
  [0;1.818181818181818;
  0;4.545454545454546];
  parameter Real C[2,4] = [1,0,0,0;0,0,1,0];
  parameter Real D[2,1] = [0;0];
  Real x[4](start=x0);
  output Real y[2];
  input Real u[1](start=u0);

  Real x_cart_x = x[1];
  Real x_cart_v = x[2];
  Real x_pendulum_phi = x[3];
  Real x_pendulum_w = x[4];
  Real u_u = u[1];
  Real y_y1 = y[1];
  Real y_y2 = y[2];
    
```

```

equation
  der(x) = A * x + B * u;
  y = C * x + D * u;
end linear_InversePendulum;
    
```

Listing 2: Linear Model of the InversePendulum at point in time 0

```

Equations (16)
=====
1 : $DER$Pcart_x$PDERcart_x = 0.0
2 : $DER$Pcart_x$PDERcart_v = 1.0
3 : $DER$Pcart_x$PDERpendulum_theta = 0.0
4 : $DER$Pcart_x$PDERpendulum_w = 0.0
5 : $DER$Ppendulum_theta$PDERcart_x = 0.0
6 : $DER$Ppendulum_theta$PDERcart_v = 0.0
7 : $DER$Ppendulum_theta$PDERpendulum_theta = 0.0
8 : $DER$Ppendulum_theta$PDERpendulum_w = 1.0
9 : (M + m) * $DER$Pcart_v$PDERcart_x +
10 : (M + m) * $DER$Pcart_v$PDERcart_v +
11 : (b + m * (1 * ($DER$Ppendulum_w$PDERcart_v * cos(pendulum_theta + pi)))) = 0.0
12 : (M + m) * $DER$Pcart_v$PDERpendulum_theta +
13 : m * (1 * ($DER$Ppendulum_w$PDERpendulum_theta * cos(pendulum_theta +
14 : pi) + (-der(pendulum_w)) * sin(pendulum_theta + pi)))) -
15 : m * (1 * (pendulum_w ^ 2.0 * cos(pendulum_theta + pi))) = 0.0
16 : (M + m) * $DER$Pcart_v$PDERpendulum_w +
17 : m * (1 * ($DER$Ppendulum_w$PDERpendulum_w * cos(pendulum_theta + pi))) -
18 : 2.0 * (m * (1 * (pendulum_w * sin(pendulum_theta + pi)))) = 0.0
19 : (i + m * l ^ 2.0) * $DER$Ppendulum_w$PDERcart_x =
20 : (-m) * (1 * ($DER$Pcart_v$PDERcart_x * cos(pendulum_theta + pi)))
21 : (i + m * l ^ 2.0) * $DER$Ppendulum_w$PDERcart_v =
22 : (-m) * (1 * ($DER$Pcart_v$PDERcart_v * cos(pendulum_theta + pi)))
23 : (i + m * l ^ 2.0) * $DER$Ppendulum_w$PDERpendulum_theta +
24 : m * (1 * (g * cos(pendulum_theta + pi))) =
25 : (-m) * (1 * ($DER$Pcart_v$PDERpendulum_theta * cos(pendulum_theta + pi)
26 : + (-der(cart_v)) * sin(pendulum_theta + pi)))
27 : (i + m * l ^ 2.0) * $DER$Ppendulum_w$PDERpendulum_w =
28 : (-m) * (1 * ($DER$Pcart_v$PDERpendulum_w * cos(pendulum_theta + pi)))
    
```

Figure 2: Equations for linear model matrices A and C

Such linear models are used in control theory for example as an observer to control the original nonlinear model [5].

4.2 Provide the analytical jacobian matrix to DASSL

For accurate, high-speed solution of DAEs as they occur in Modelica (see equation (2)) Petzold's Fortran-based DASSL(Differential-Algebraic System Solver) is the most widely used sequential code for solving such DAEs. After all, the DASSL implementation uses the following equation [8]

$$h(t, x, \hat{\alpha}x + \beta) = 0,$$

where $\hat{\alpha}$ is a constant which changes whenever the step size or the order changes, β is a vector which depends on the solution at past times and $t, x, \hat{\alpha}, \beta$ are evaluated at t_n . This equation is solved in DASSL by a modified version of Newton's method,

$$x^{m+1} = y^m - cj \left(\frac{\partial h}{\partial x} + cj * \frac{\partial h}{\partial \hat{x}} \right)^{-1} h(t, x, \hat{\alpha}x + \beta).$$

The iteration matrix

$$M = \frac{\partial h}{\partial x} + cj * \frac{\partial h}{\partial \hat{x}}$$

is computed and factored, and is then used for as many time steps as possible.

By default DASSL calculates the iteration matrix M by the means of numerical finite differentiation. However, it is also possible to equip DASSL with an user-specific routine that provides the symbolically calculated iteration matrix M . On one hand, the symbolically calculated values are more accurate and on the other hand, it is faster to evaluate the symbolical formulas.

5 Conclusion and Future Work

The successful implementation of symbolically generated partial derivatives for the corresponding Jacobian matrices using automatic differentiation methods in the OpenModelica Compiler has been demonstrated. The new feature supports all Modelica language elements and Modelica models already handled by OMC. The corresponding symbolic derivative module has been validated by creating linear models for non-linear Modelica models. Furthermore, providing the analytically determined Jacobian matrix to DASSL, leads to a faster simulation of the model. In addition to this, this implemented methods offer a variety of different application fields (i.e. parameter identification, sensitivity analysis, uncertainty calculation,

inline-integration methods, model reduction, optimization ...).

In future it is possible to improve this module in two directions: First this module could be made accessible for the user. The user could select some functions of equations (3) and some depended variables. Thus the user can decide which symbolic matrices is wanted in the simulation program. The second direction could be to generate directly a Modelica model with the symbolic derivative expressions. With this approach the symbolic matrices could be made accessible during simulation in a way so that the updated version can always be used for controlling or optimization processes.

Other initiatives aim to extend the Functional Mock-up Interface for model exchange [2] to support the evaluation of sparse Jacobians. This work can easily be adapted to provide the required calculations.

Acknowledgments

The German Ministry BMBF has partially funded this work (BMBF Förderkennzeichen: 01IS09029C) within the ITEA2 project OPEN-PROD (<http://www.openprod.org>).

References

- [1] Elsheikh A., Noack S. and Wiechert W.: Sensitivity analysis of Modelica applications via automatic differentiation, 6th International Modelica Conference, Bielefeld, 2008.
- [2] MODELISAR: Functional Mock-up Interface for Model Exchange, http://modelisar.org/specifications/FMI_for_ModelExchange_v1.0.pdf, Januar 2010.
- [3] Fritzson P. et. al.: OpenModelica System Documentation, PELAB, Department of Computer and Information, Linköpings universitet, 2010.
- [4] Imsland L., Kittilsen P. and Schei T.: Using Modelica models in the real time dynamic optimization - gradient computation, Proceedings 7th Modelica Conference, Como, 2009.
- [5] Lunze, J.: Regelungstechnik 2 – Beobachterentwurf, Springer-Lehrbuch, Springer Berlin Heidelberg, 2010.

- [6] Modelica Association: Modelica – A unified Object-oriented Language for Physical Systems Modeling Language Specification – Version 3.2, 2010.
- [7] Otter M.: Objektorientierte Modellierung Physikalischer Systeme (Teil 4) Transformationsalgorithmen, Automatisierungstechnik, Oldenbourg Verlag München, 1999.
- [8] Petzold L. R.: A Description of DASSL: A Differential/Algebraic System Solver, Sandia National Laboratories Livermore, 1982.
- [9] Rall L.B.: Automatic differentiation: Techniques and applications, vol. 120 of Lecture Notes in Computer Science, Springer, 1981.

Cyber-Physical Systems Modeling and Simulation with Modelica

Dan Henriksson and Hilding Elmqvist
 Dassault Systèmes AB, Ideon Science Park, Lund, Sweden
 Dan.Henriksson@3ds.com, Hilding.Elmqvist@3ds.com

Abstract

This paper introduces the area of Cyber-Physical Systems (CPS) and describes the relation to Modelica and Modelica-based tools. Special aspects of CPS applications that should make Modelica well suited for their modeling and simulation are highlighted.

Recent Modelica developments facilitating integrated model-based system development applicable to CPS are presented. Especially, it is shown how detailed timing simulations, involving both real-time task scheduling and network communication, are realized in Modelica. A robot example is used to demonstrate the new CPS simulation features.

Keywords: Cyber-physical systems, Modelica, Model-based development, Timing simulation.

1 Introduction

During the last few years the field of Cyber-Physical Systems (CPS) has emerged, mainly in the US, as one of the most important academic and industrial research topics for the future. CPS focus on the interaction between computing/communication and the physical world. The following definition of CPS is taken from Wikipedia [1]

*“A cyber-physical system (CPS) is a system featuring a tight combination of, and coordination between, the system’s **computational** and **physical** elements.”*

Embedded computing devices controlling physical processes are today found in many diverse application areas, including automotive, aerospace, energy, and telecommunications. These traditional embedded systems can be viewed as a sub-set of the wider CPS definition. Typical CPS applications tend to be large-scale distributed systems built up from networks of computing/communication devices and possibly also distributed physical plants, such as in power grids. The complexity and scale inherent to

CPS should make Modelica well suited for modeling and simulation of these systems.

The US National Science Foundation (NSF) has identified cyber-physical systems as a key area of research and several workshops have been organized on various aspects of CPS. The international multi-conference, CPSWeek, has been organized since 2008 and *The Cyber-Physical Systems Summit* was held in conjunction with the first CPSWeek in St Louis, Missouri. This resulted in a report [2] that outlines the CPS field, including important applications and future research challenges. One of the observations in the report is that traditionally

“... the fields of computer science and control theory have remained largely separate, both technically and culturally.”

To be able to build complex applications that successfully combine computational elements with elements from the physical world, it is important to have a new, integrated approach to the design. Modelica and Modelica-based tools should comprise good environments for such integrated model-based development. The report goes on to state that

*“On the one hand, computer engineers and scientists do not know how to translate requirements for physical systems, such as **stability**, into computational requirements on performance, power consumption, etc.”*

*“On the other hand, control and signal processing theory abstract computers largely as infallible numerical devices. This simplification **ignores** many important aspects of computing, such as increasingly larger **timing variance** due to caches and energy management and increasingly higher software error rates caused by complexity.”*

Again, a unified modeling environment supporting integrated simulation of all these aspects is needed. Finally, it is concluded that

*“Simplifying assumptions are also made about communications. Initial designs **assume zero-loss, zero-delay communications**, while neither occur in the wireless, low-power, shared, rapidly changing systems used in most CPS. The viability of future CPS must also address **noise in measurements, inaccuracies in actuation, disturbances from the environment, and faults and failures in the computational process in a coherent, unified framework.**”*

The motivation for this paper is to introduce the CPS field to the Modelica community and to describe how recent Modelica developments will facilitate integrated modeling and simulation of the cyber and physical parts of complex, hierarchical systems. The presented framework allows simulation of non-ideal behavior, such as delays, noise, and quantization, as well as detailed timing simulations of real-time task scheduling and network communication. A robotics system will be used as demonstrator.

1.1 Related Work

The timing simulations presented in this paper are based on the TrueTime simulator [3, 4] developed at the Department of Automatic Control, Lund University, Sweden. The original version of TrueTime is based on MATLAB/Simulink [5] and the first Modelica-based version of TrueTime was presented in [6]. The version in [6], however, only dealt with the network simulation part of TrueTime, whereas the timing simulations presented in this paper also include simulation of real-time task scheduling in Modelica. In this paper, it is also shown how the timing simulations are integrated in a unified Modelica framework for systems configuration and simulation.

Most of the material presented in this paper was first prepared for a Modelica tutorial organized by the authors at the 3rd international CPSWeek multi-conference in Stockholm, April 2010.

1.2 Outline

The rest of this paper is outlined as follows. Section 2 continues the discussion on Modelica in the context of CPS and presents the recent development of Modelica for embedded systems with focus on features for simulation of non-ideal effects in controllers. Detailed simulation of controller timing variations in this new Modelica framework is introduced in Section 3. The cyber-physical simulation features are demonstrated on a robot example in Section 4. Finally, Section 5 gives the conclusions and directions for future work.

2 Modelica and CPS

The Cyber-Physical Systems Summit identified a series of applications that should give substantial societal impact while also presenting significant challenges. Three of these applications were developed into main grand challenges for CPS; *future distributed energy systems, future transportation systems, and next-generation healthcare systems.*

The first two challenges represent areas where the physical modeling capabilities of Modelica already are very strong, and medical and biological systems (third challenge) also contain physical elements well suited for equation-based modeling. Other applications of CPS include autonomous automotive systems, automatic pilot avionics, and distributed robotics.

There are, thus, many applications of CPS that require detailed physical modeling. However, since the physical modeling capabilities of Modelica are well established, this paper focuses on recent Modelica features that will allow modeling and co-simulation of the “cyber” aspects of CPS together with the physical models. The following aspects need to be addressed:

- The scale and complexity of CPS require compositional methods for integrated design and modeling.
- Most CPS applications are built up from controlled sub-systems, systems-of-systems, with local interaction between physical systems and controllers affecting global performance.
- Distributed sensing, actuation, and control needs to be modeled and simulated.
- The interfaces between the cyber and physical needs to be identified and properties of the interfaces should be easy to specify.
- New hybrid models may be needed to completely model all aspects of CPS.

2.1 Modelica for Embedded Systems

A first step towards accomplishing the aspects above in Modelica has been performed in the ITEA2 project EUROSYSLIB [7]. Language constructs were added to support embedded systems modeling and configuration using Modelica. A support library, *Modelica EmbeddedSystems*, was developed as a user-friendly interface to the new language constructs. The new language constructs and a preliminary version of this library were presented in [8]. The effort to improve the cyber aspects of Modelica

has continued in the ITEA2 MODELISAR [9] project, focusing on timing simulations and connections to the AUTOSAR [10] standard.

The new Modelica language features for embedded systems exhibit two important properties strongly related to the CPS aspects outlined above.

1. In complex models, constructed as a hierarchy of controlled sub-systems, a Modelica translator will be able to automatically deduce which parts that are truly physical, and which parts that are digital controllers (“cyber”).
2. Special *communication blocks* are inserted in the interfaces between the cyber and physical parts. These contain replacable components, which makes it possible to easily define different properties (such as timing variations, noise, and inaccuracies) without modifying the structure of the original model.

2.2 Simulated Communication

The Modelica_EmbeddedSystems library contains special communication block implementations allowing simulation of non-ideal communication between controllers and plants. The effects that can be simulated are; sampling, computational delay (fraction of sample period), communication delay (unbounded), measurement noise, signal limitations, and quantization.

These effects are easily configured in the parameter dialog shown in Figure 1 below. The dialog is accessed from the communication blocks that make up the interfaces between controller and plant components of the controlled sub-systems.

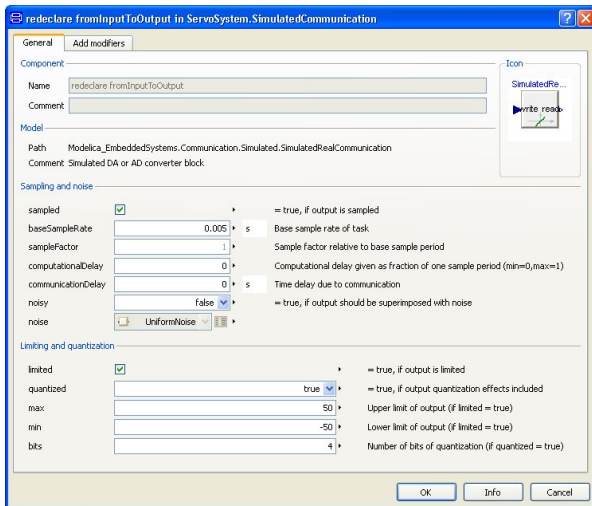


Figure 1: Parameter dialog for the simulated communication block.

3 Timing Simulations

The simulated communication described above has been extended to enable more realistic simulation of timing effects. For example, latencies due to computation and/or communication are rarely constant but will vary as a result of the chosen task scheduling policy and communication protocol. These variations, called jitter, may also affect the actual sampling interval and could have a degrading effect on the performance of closed-loop control systems [3].

The developed timing simulations mimic the temporal behavior of real-time operating systems (RTOS) executing tasks in embedded processors and transmission of messages over communication networks. Aspects that can be simulated include latencies and jitter due to preemption and real-time task scheduling, queuing in the sending and receiving network nodes, latencies due to signal transmission, and latencies and jitter due to collisions and media access control (MAC) policies.

The simulations focus on control networks, i.e., networks sending relatively small packets regularly with tight real-time constraints [11]. The following real-time network protocols [12,13,14,15] are currently supported; Ethernet, CAN, TTCAN, FlexRay, and static time-triggered communication (TDMA).

3.1 Simulation Configuration

Task and subtask configuration records, provided in the Modelica_EmbeddedSystems library, are used to specify the real-time attributes (periods, offsets, priorities, and deadlines) of the simulated controller tasks. In addition, special kernel and network simulator models have been developed in Modelica to realize the timing simulators. These blocks use Modelica_EmbeddedSystems configuration records as input as shown for the kernel simulator in Figure 2.

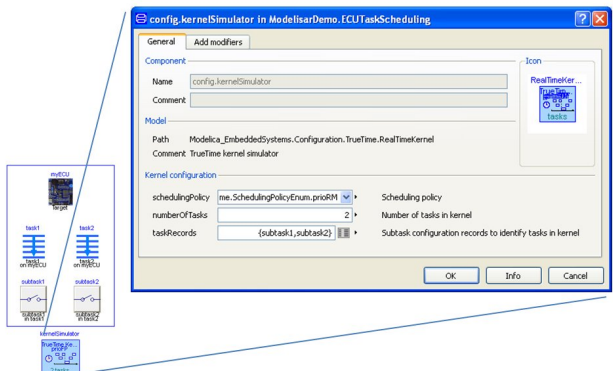


Figure 2: Configuration of the kernel simulator.

The kernel simulator is configured by providing the desired scheduling policy, the number of simulated tasks, and an array of subtask records. The kernel simulator currently supports the following scheduling policies [16]:

- Fixed-priority scheduling
- Rate-monotonic scheduling
- Deadline-monotonic scheduling
- Earliest-deadline-first scheduling

The network configuration consists of general parameters (MAC policy, number of frames, data rate, minimum frame size, and loss probability) and policy-specific parameters. Policy-specific parameters needs to be specified for TDMA (Time Division Multiple Access), TTCAN, and FlexRay and consist of slot sizes and static communication schedules (represented as arrays of frame identifiers). In addition, FlexRay configuration also needs a communication schedule for its dynamic segment and a network idle time (in bits). The complete parameter dialog is shown in Figure 3.

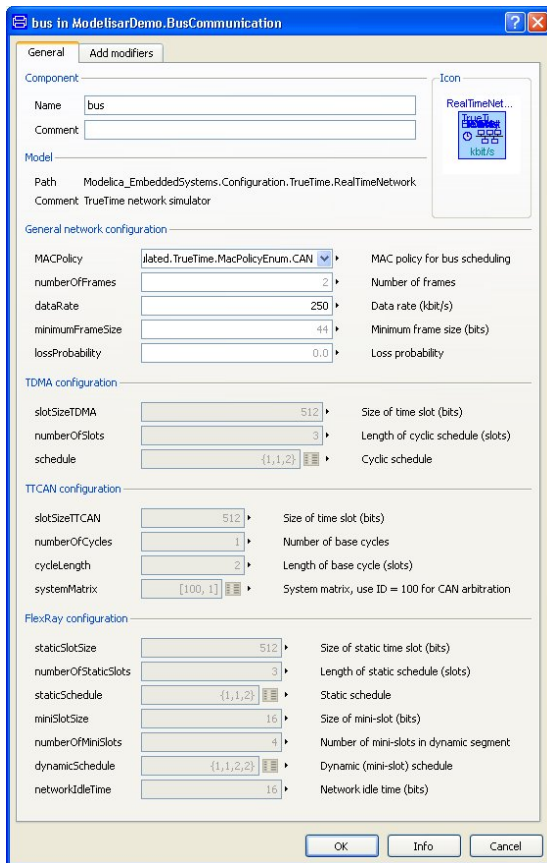


Figure 3: Configuration of the network simulator.

3.2 Communication

Having configured the kernel and/or network simulators, it is also required to configure individual signals in the model to use the timing simulators. This is done using the parameter dialog of the communication blocks. This configuration involves specifying the execution times of tasks, sizes and priorities of network frames, and references to the kernel and network simulators associated with the signal, see Figure 4.

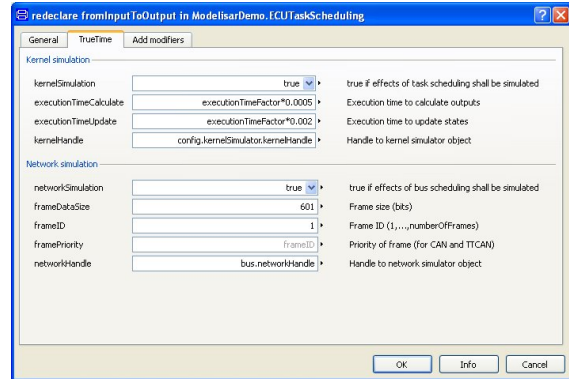


Figure 4: Configuration of individual signals.

3.3 Simulation Output

The kernel and network simulator models generate traces according to Figure 5. For each task in a simulated kernel, a task activation graph is produced. The graph has three levels, corresponding to idle, preempted by a higher-priority task, and running. Similarly for network frames, it is shown when a frame is being transmitted and when it is waiting because another transmission is using the bus. These graphs can be used to identify and verify timing properties.

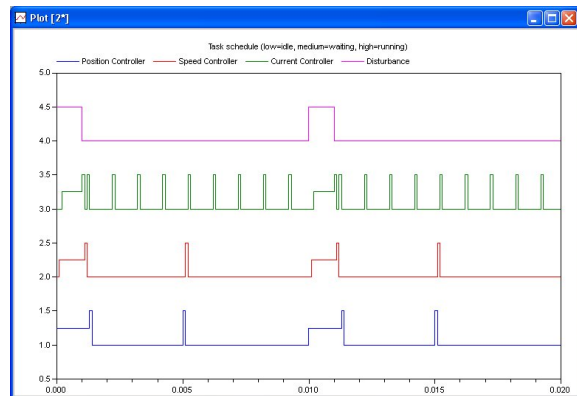


Figure 5: Example of timing graph.

3.4 Modelica Implementation

The original TrueTime simulator is a Simulink S-function implemented in C++. The Modelica implementation uses the C external function interface to call the simulator at appropriate time instants.

The simulator is event-based and triggered based on internal and external events. Each time the simulated real-time kernel or network is executed, it computes the time for next internal, scheduled, event. The Modelica wrapper of the simulator then makes sure to generate a time event at the scheduled event. Examples of scheduled events are the completion of a network transmission or a real-time task finishing execution.

In between scheduled events, the kernel or network simulators could also be triggered by external events. External events include sending of a network message and triggering of task jobs.

3.5 Use Cases and Possible Extensions

The timing simulations can be used both to generate and to verify timing requirements of controller components. For example, with the new FMI [17] interface for model exchange and co-simulation it will be straight-forward to import a controller component, hook it up to the timing simulator and co-simulate it with its physical environment.

The current timing simulations assume that accurate execution time estimates are available for all tasks. A future extension will be to integrate the task simulations with tools for execution-time analysis. It will also be investigated how to integrate the timing simulations and task configurations with the timing model available in the latest AUTOSAR specification, version 4.0.

4 Simulation Examples

This section will demonstrate the simulation capabilities on a simple robotics example. It will first be shown how to simulate simple non-ideal behavior using the various options in the simulated communication parameter dialog of Figure 1. Then it will be demonstrated how to perform more detailed timing simulations including high-priority tasks and network transmissions.

4.1 Simulation Model

The simulation model, shown in Figure 6, contains three main components on the top level, a path plan-

ner, a controlled servo, and the mechanical model of a one-armed robot.

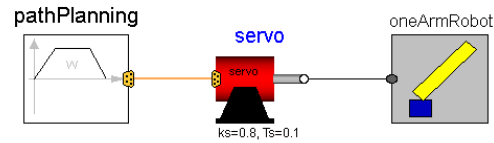


Figure 6: Simulation example.

The servo component contains a DC motor connected to a non-linear gear controlled in a three-level cascade with control loops for the motor angle, angular velocity, and motor current. Figure 7 shows the internals of the servo component. The controller component contains the angle and velocity P- and PI-controllers, and it is shown how communication blocks have been inserted at the interfaces between the controllers and the physical plant.

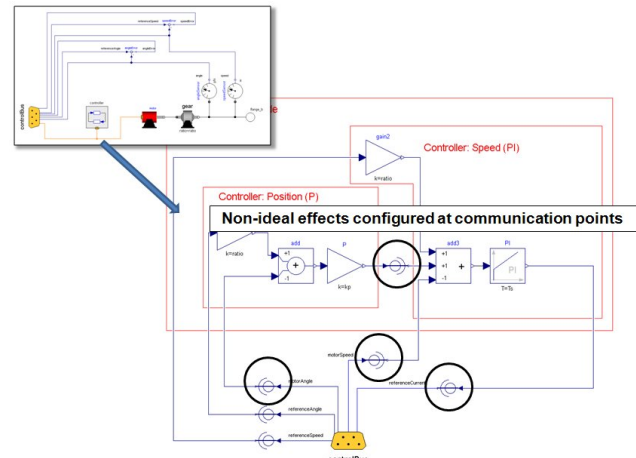


Figure 7: Controlled servo extended with communication blocks.

4.2 Simulating Non-ideal Communication

The simulations below will consider a step change of ten degrees in the angle reference and the plots will show the angle, angular velocity and the controller output of the velocity loop (used as reference to the current controller). The ideal continuous case is included for reference in the simulation graphs.

We will first consider the effects of using sampled instead of continuous controllers. This is achieved by marking the checkbox *sampled* and selecting a sample rate (see Figure 1). In this case the sample rate is deliberately chosen too slow in relation to the closed-loop dynamics, and the results of

that simulation are shown in Figure 8. The effects of too slow sampling are clearly seen in the velocity signal and in the control signal.

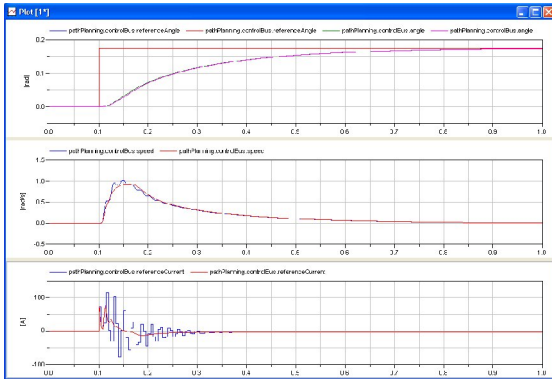


Figure 8: Simulation of sampled controller.

Next we will impair the stability properties of the system even more by also adding delay to the control signal computations. The results are shown in Figure 9, and in this case the performance of the system deteriorates to the point where the system starts to oscillate.

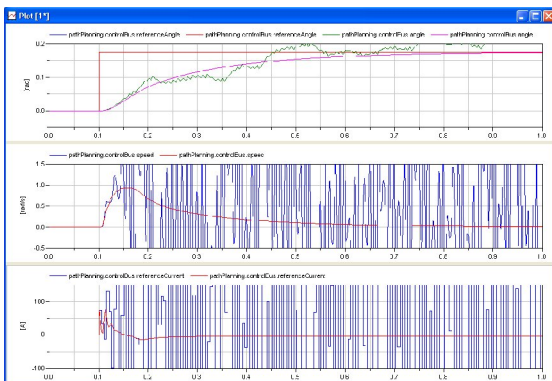


Figure 9: Sampled controller with delay.

Figure 10 shows the effects of adding measurement noise to the velocity measurement and it is seen how the noise is propagated and amplified by the velocity controller. Finally, Figure 11 demonstrates the possibility to simulate limitations on signals and signal quantization. In this case, the velocity controller output is limited between -50 and 50 with a very coarse resolution of only four bits.

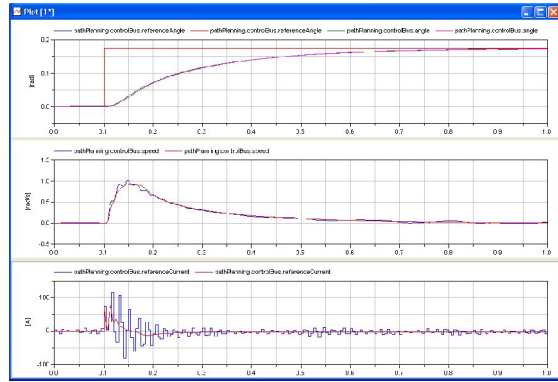


Figure 10: Sampled controller with measurement noise.

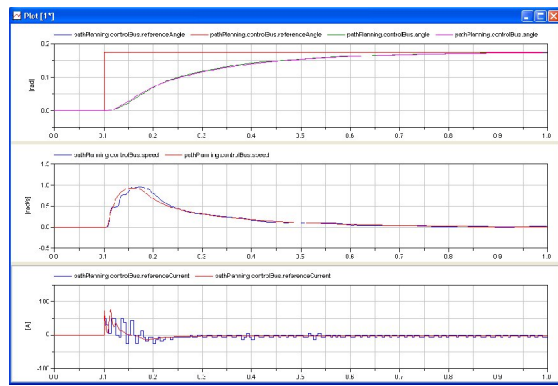


Figure 11: Sampled controller with output quantization.

4.3 Timing Simulations

For the detailed timing simulations, we extend the robot model with a configuration component (including timing simulators and records to specify task attributes) and disturbing computations and communication according to Figure 12.

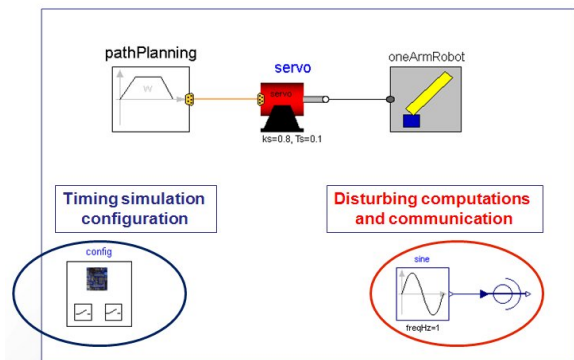


Figure 12: Extended robot model for timing simulation.

We will first examine the effects of real-time task scheduling. The simulated real-time kernel contains four tasks, representing the angle controller, velocity controller, current controller, and a disturbance task. The angle and velocity controller tasks have a period of 5 ms, whereas the current controller is executed with a period of 1 ms. The disturbance task runs on a 10 ms period. The simulated execution times are 100 μ s for the controller tasks and 1 ms for the disturbance task.

Figure 13 shows the result when the disturbance task is given the highest priority. In this case, the disturbance task interferes with the controller tasks every 10th sample and the timing variations, as shown in the task schedule in Figure 14, causes the performance to deteriorate.

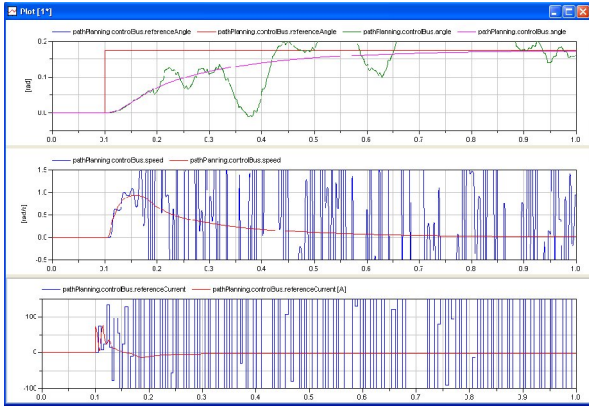


Figure 13: Simulation performance with high-priority disturbance task.

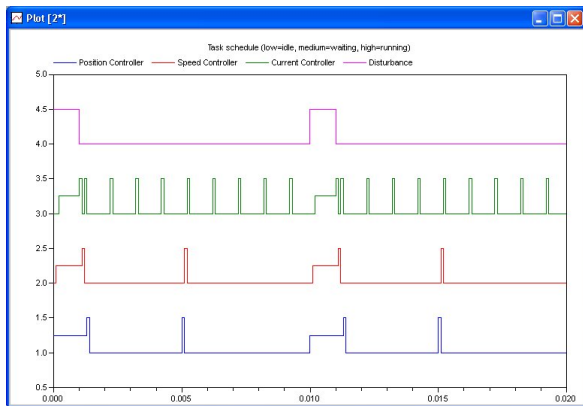


Figure 14: Task schedule with high-priority disturbance task.

By changing the scheduling policy to rate-monotonic scheduling (the shorter period, the higher priority), the controller task will automatically get higher priority than the slow disturbance task. In this

case, the control performance is the same as for the 5 ms sampled simulation shown in Figure 8. The task schedule for this simulation is shown in Figure 15.

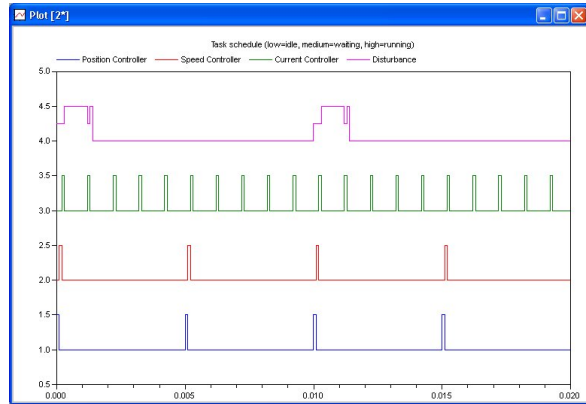


Figure 15: Task schedule when using rate-monotonic scheduling.

Finally, we will consider also network communication. In this simulation we assume that the control computer and the actuator reside on different nodes. The control signal computed by the inner controller (the current controller) then needs to be sent over the network. We also assume that the disturbance node sends high-priority traffic on the network.

We will compare two different network policies, CAN and FlexRay. CAN schedules frames according to priority, whereas FlexRay has a static segment where frames get exclusive network access during certain time slots. Figures 16 and 17 show the different transmission schedules obtained for CAN and FlexRay. For CAN, the disturbance frames interfere with the control signal transmissions. In the case of FlexRay, the controller frames are transmitted in the static segment and are then not affected by the disturbance frames.

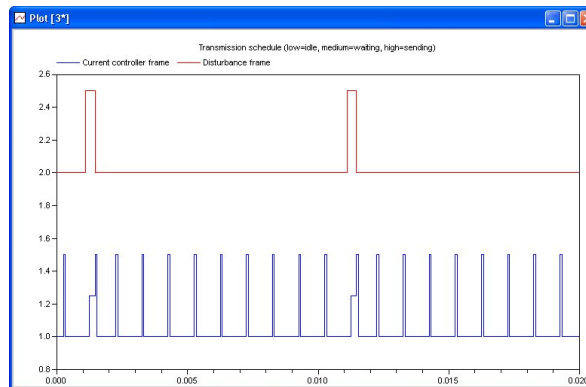


Figure 16: Transmission schedule using CAN configuration.

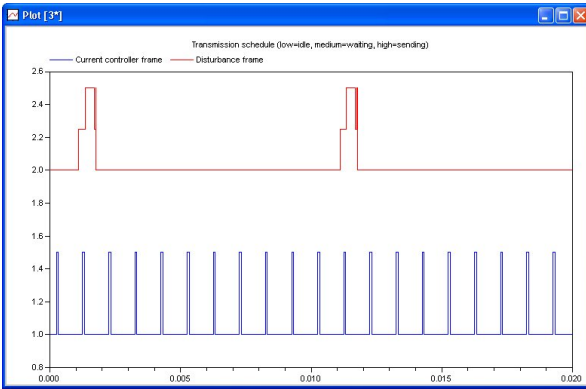


Figure 17: Transmission schedule using FlexRay configuration.

5 Conclusions

This paper has introduced the field of cyber-physical systems (CPS) and has outlined some first steps in supporting CPS modeling and simulation using Modelica. It was shown how non-ideal controller behavior could be conveniently configured and simulated using the Modelica_EmbeddedSystems library. The simulated phenomena included sampling, delays, noise, limited signals, and quantization.

It was further demonstrated how detailed timing simulations based on the TrueTime simulator has been integrated in the Modelica_EmbeddedSystems framework. The presented timing simulations included both network simulations and simulations of task scheduling in real-time kernels. A robot example was used as a demonstrator of the presented simulation features.

5.1 Future Work

Future work will mainly focus on three areas; integration with tools for execution-time analysis, connections to the AUTOSAR 4.0 timing model, and extensions to support simulation of wireless network protocols. The first two areas were elaborated in Section 3.5.

Most CPS applications rely on wireless sensor networks in order to collect measurements that are used to close feedback loops. Thus, modeling of wireless network protocols and simulation of aspects related to this, such as signal attenuation, fading, packet losses, power consumption, etc is important in order to fully support CPS modeling and simulation. The Modelica version of TrueTime presented in [6] supports the wireless network protocols IEEE 802.11b/g (WLAN) and 802.15.4 (ZigBee).

5.2 Acknowledgements

We thank the Swedish funding agency VINNOVA for partial funding of this work within the ITEA2 project MODELISAR (2008-02291).

References

- [1] http://en.wikipedia.org/wiki/Cyber-physical_system
- [2] <http://varma.ece.cmu.edu/Summit/>
- [3] Cervin A., Henriksson D., Lincoln B., Eker J., and Årzén K.-E. *How Does Control Timing Affect Performance? Analysis and Simulation of Timing Using Jitterbug and TrueTime*. IEEE Control Systems Magazine, 23:3, June, 2003.
- [4] <http://www.control.lth.se/truetime>
- [5] <http://www.mathworks.com/>
- [6] Reuterswård P., Åkesson J., Cervin A., and Årzén K.-E. *TrueTime Network – A Network Simulation Library for Modelica*. In Proceedings of the 7th Modelica Conference, Como, Italy, September, 2009.
- [7] <http://www.eurosyslib.com/>
- [8] Elmquist 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.
- [9] <http://www.modelisar.org/>
- [10] <http://www.autosar.org/>
- [11] Lian F.-L., Moyne J., and Tilbury D. *Network Protocols for Networked Control Systems*. In Handbook of Networked and Embedded Control Systems (Hristu-Varsakelis and Levine Eds.), 2005.
- [12] Paret, D. *Multiplexed Networks for Embedded Systems, CAN, LIN, FlexRay, Safe-by-Wire...*, Wiley, ISBN 978-0-470-03416-3, 2007.
- [13] <http://www.can-cia.org>
- [14] <http://www.can-cia.org/index.php?id=521>
- [15] <http://www.flexray.com>
- [16] Liu J. W. S. *Real-Time Systems*. Prentice Hall, 2000.
- [17] <http://www.functional-mockup-interface.org>

Bootstrapping a Modelica Compiler aiming at Modelica 4

Martin Sjölund, Peter Fritzson, Adrian Pop

PELAB – Programming Environment Lab, Dept. Computer Science
Linköping University, SE-581 83 Linköping, Sweden
{ martin.sjolund, peter.fritzson, adrian.pop }@liu.se

Abstract

What does it mean to bootstrap a compiler, and why do it? This paper reports on the first bootstrapping (i.e., a compiler can compile itself) of a full-scale EOO (Equation-based Object-Oriented) modeling language such as Modelica. The Modelica language has been modeled/implemented in the OpenModelica compiler (OMC) using an extended version of Modelica called MetaModelica. OMC models the MetaModelica language and is now compiling itself with good performance. Benefits include a more extensible maintainable compiler, also making it easier to add functionality such as debugging support.

This work is in line with the recently started Modelica 4 design effort which includes moving implementation of language features from the compiler to a Modelica Core library, allowing compilers to become smaller while increasing correctness and portability.

A number of language constructs discussed for Modelica 4 are already supported in some form by the bootstrapped compiler. Future work includes adapting language constructs according to the Modelica 4 design effort and extracting and restructuring parts of the Modelica implementation from the OMC compiler to instead reside in a Modelica Core library, making the compiler smaller and more extensible.

Keywords: Compilation, Modelica, MetaModelica, meta-programming, metamodeling, modeling, simulation

1 Introduction

Since user requirements on the usage of models grow over time, and the scope of modeling domains increase, the demands on the Modelica modeling language and corresponding tools increase. This has caused the Modelica language and model compilers to become increasingly large and complex.

One approach to manage this increasing complexity used by several functional languages (Section 6) is to define a number of language features in libraries rather

than in the compiler itself. After several years of discussion and a number of language prototypes, e.g., [3][8][9][23][30][32][37], this approach finally became part of the new Modelica 4 effort started during the 67th Modelica design meeting [22].

1.1 Modeling Language Constructs

The Modelica specification and modeling language was originally developed as an object-oriented declarative equation-based specification formalism for mathematical modeling of complex systems, in particular physical systems.

However, it turns out that with some minor extensions, the Modelica language is well suited for another modeling task, namely modeling of the semantics, i.e., the meaning, of programming language constructs. Since modeling of programming languages is often known as meta-modeling, we use the name MetaModelica [8] [11][12] [30] for this slightly extended Modelica.

The semantics of a language construct can usually be modeled in terms of combinations of more primitive builtin constructs. One example of primitive builtin operations are the integer arithmetic operators. These primitives are combined using inference and pattern-matching mechanisms in the specification language.

Well-known language specification formalisms such as Structured Operational Semantics/Natural Semantics [28] are also declarative equation-based formalisms. These fit well into the style of the MetaModelica specification language, which explains why Modelica with some minor extensions is well-suited as a language specification formalism. However, only an extended subset of Modelica here called MetaModelica is needed for language specification since many parts of the language designed for physical system modeling are not used at all, or very little, for language specification.

Another great benefit of using and extending Modelica in this direction is that the language becomes suitable for meta-programming and meta-modeling. This means that Modelica can be used for specification and

transformation of models and programs, including transforming and combining Modelica models into other (lower-level) Modelica models, i.e., a kind of compilation.

Figure 1 shows typical translation stages in a Modelica compiler.

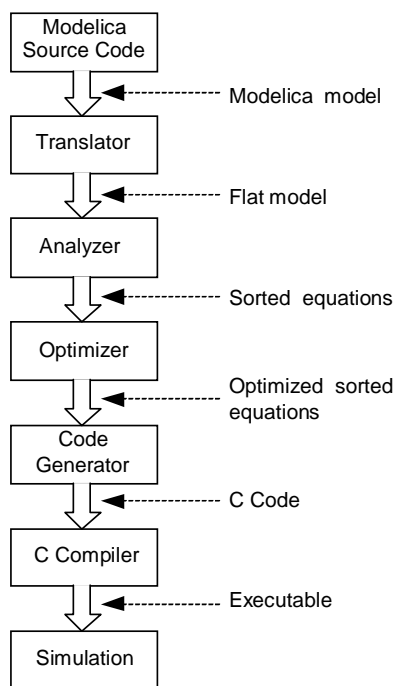


Figure 1. The typical stages of translating and executing a Modelica model.

2 Vision – Extensible Tools

Traditionally, a model compiler performs the task of translating a model into executable code, which then is executed during simulation of the model. Thus, the symbolic translation step is followed by an execution step, a simulation, which often involves large-scale numeric computations.

However, as requirements on the usage of models grow, and the scope of modeling domains increases, the demands on the modeling language and corresponding tools increase. This causes the model compiler to become large and complex.

Moreover, the modeling community needs not only tools for simulation but also languages and tools to create, query, manipulate, and compose equation-based models. Additional examples are optimization of models, parallelization of models, checking and configuration of models.

If all this functionality is added to the model compiler, it tends to become large and complex.

An alternative idea already mentioned in Section 1 is to add features to the modeling language defined in

library packages that can contain model analysis and translation features that therefore are not required in the model compiler. An example is a PDE discretization scheme that could be expressed in the modeling language itself as part of a PDE package instead of being added internally to the model compiler.

2.1 Why Bootstrapping?

As mentioned, bootstrapping means that a compiler can compile itself. What are the pros and cons of bootstrapping?

- The implemented language becomes well tested, since the developers are using it on a large application (the compiler).
- The developers are motivated to make a high quality implementation, since they are using it themselves.
- The developers are motivated to create a good development environment, since they are using it themselves.
- There is also a negative factor: since the tool must be able to build itself, there is more work to do significant changes to it – the implementation must be good enough to be useable.

2.2 The Stages of Bootstrapping OMC

The bootstrapping of the OpenModelica Compiler (OMC) has been a 5-year effort, consisting of the following stages:

1. Design of an early MetaModelica language version [8] as an extended subset of Modelica, spring 2005.
2. Implementation of a MetaModelica Compiler (MMC) by adding a new compiler frontend to the old RML compiler [13] [28], translating MetaModelica into RML intermediate form, spring-fall 2005.
3. Automatically translating the whole OpenModelica compiler, 60 000 lines, from RML to MetaModelica.
4. In parallel, developing a new Eclipse plugin, MDT (Modelica Development Tooling), for Modelica and MetaModelica [29][31], including both browsing, debugging, semantic context-sensitive information, etc., 2005-2006.
5. Switching to using this MetaModelica 1.0 (an extended subset of Modelica), the MMC compiler, and the new MDT Eclipse plugin for the OpenModelica compiler development, 3-4 full-time developers. This version 1.0 of MetaModelica is described in [10][11]. Fall 2006.
6. Preliminary implementation of pattern-matching [34] and exception handling [31] in the OpenMo-

delica compiler, to enable future bootstrapping. Spring-fall 2008.

7. Continuation of the work on better support for pattern-matching compilation, support for lists, tuples, records, etc. in OpenModelica, as part of metamodeling support in the OMC Java interface [33] Spring-fall 2009.
8. Implementation of function arguments to functions (used in MetaModelica), also in OpenModelica [1]. This also became part of the Modelica 3.2 standard [20]. Fall 2009, spring 2010.
9. The current work on finalizing the bootstrapping reported in this paper. The bootstrapped compiler supports MetaModelica 2.0, which includes both standard Modelica as well as further improved MetaModelica extensions aiming at Modelica 4 support. Fall 2010, spring 2011.
10. Further Adding, enhancing, and redesigning MetaModelica language features [12] based on usage experience, the Modelica 4 design effort, and inspiration from functional languages and languages such as Scala [26]. Refactoring parts of the compiler to use the enhanced features.

3 MetaModelica

As already mentioned, MetaModelica provides language extensions for language modeling and model transformations. The basic language extensions are briefly described here. Some features are defined in libraries.

3.1 Pattern Matching

Pattern matching expressions [34] may occur where expressions can be used in Modelica code. There are two kinds of match-expressions in MetaModelica, using the `match` or `matchcontinue` keywords. The syntax can be described (approximately) as follows.

```

matchcontinue (<var-list>)
  local
    <var-decls>
    ...
  case (<pat-expr>)
    equation
      <equations>
    then <expr>;
  ...
end matchcontinue;

```

Only local, time-independent equations may occur inside a pattern matching expression and this must be checked by the semantic phase of the compiler. The difference between a pattern matching expression with the keyword `match` and a pattern matching expression with the keyword `matchcontinue` is in the fail semantics.

The `<pat-expr>` expression is a sequence of patterns. A pattern may be:

- A wildcard pattern, denoted `_`.
- A variable, such as `x`.
- A constant literal of built-in type such as `7` or `true`.
- A variable binding pattern of the form `x as pat`.
- A constructor pattern of the form `C(pat1,...,patN)`, where `C` is a record identifier and `pat1,...,patN` are patterns. The arguments of `C` may be named (for instance `field1=pat1`) or positional but a mixture is not allowed. We may also have constructor patterns with zero arguments (constants).

3.1.1 Semantics

The semantics of a pattern matching expression is as follows: If the input variables match the pattern-expression in a case-clause, then the equations in this case-clause will be executed and the `matchcontinue` expression will return the value of the corresponding then-expression. The variables declared in the uppermost variable declaration section can be used (as local instantiations) in all case-clauses. The local variables declared in a case-clause may be used in the corresponding pattern and in the rest of the case-clause. The matching of patterns works as follows given a variable `v`.

- A wildcard pattern, `_`, will succeed matching anything.
- A variable, `x`, will be bound to the value of `v`.
- A constant literal of built-in type will be matched against `v`.
- A variable binding pattern of the form `x as pat`: If the match of `pat` succeeds then `x` will be bound to the value of `v`.
- A constructor pattern of the form `C(pat1,...,patN)`: `v` will be matched against `C` and the subpatterns will be matched (recursively) against parts of `v`.

3.1.2 Pattern Matching Fail Semantics

If a case-clause fails in an expression with the keyword `matchcontinue` then an attempt to match the subsequent case-clause will take place. If we have an expression with the keyword `match`, however, then the whole expression will fail if there is a failure in one of the case-clauses. We will henceforth only deal with `matchcontinue` expressions.

3.2 Data Types

`List`, `Tuple` and `Option` are algebraic data types that are common in many languages used for meta-programming and symbolic programming. The `union-`

`type` is a recursive type required to represent trees and directed acyclic graphs.

3.2.1 Lists

The following operations allow creation of lists and addition of new elements in front of lists in a declarative way. Extracting elements is done through pattern-matching in match-expressions shown earlier.

- `List(e11,e12,e13, ...)` creates a list of elements of identical type. Examples: `List()` – the empty list, `List(2,3,4)` – a list of integers.
- `{}` – denotes an empty reference to a list.
- `cons` – the call `cons(element, lst)` adds an element in front of the list `lst` and returns the resulting list. Also available as a new built-in operator `::` (coloncolon), e.g. used as in: `element::lst`.

Types of lists and list variables can be specified as follows:

```
type RealList = List<Real>;
```

or directly in a declaration of a variable `rlist` that denotes a list of real numbers:

```
List<Real> rlist;
```

3.2.2 Tuples

Tuples can be viewed as instances of anonymous records. The syntax is a parenthesized list. The same syntax is used in extended Modelica presented here, and is in fact already present in standard Modelica as a receiver of values for functions returning multiple results.

- An example of a tuple literal:
`(a, b, "cc")`
- A tuple with a single element has a comma in order to have different syntax compared to a parenthesized expression: `(a,)`
- A tuple can be seen as being returned from a function with multiple results in standard Modelica:
`(a,b,c) := foo(x, 2, 3, 5);`
- Access of field values in tuples is achieved via pattern-matching or dot-notation, `tupval.fieldnr`, analogous to `recval.fieldname` for ordinary record values. For example, accessing the second value in `tup`:
`tup.2`

The main reason to introduce tuples is for convenience of notation. You can use them directly without explicit declaration. Tuples using this syntax are already present in the major functional programming languages.

A tuple will of course also have a type. When tuple variable types are needed, they can for example be declared using the following notation:

```
type VarBND = Tuple<Ident, Integer>;
```

or directly in a declaration of a variable `bnd`:

```
Tuple<Ident, Integer> bnd;
```

3.2.3 Option Types

Option types have been introduced in MetaModelica to provide a type-safe way of representing the common situation where a data item is optionally present in a data structure. In C-like languages this is often represented by NULL pointers, which are not type-safe and may cause program crashes.

- `NONE()` represents no data present
- `SOME(e)` represents `e` present

The option type is declared similar to the list type.

```
type MaybeResult = Option<Result>;
```

3.2.4 MetaModelica Array Types

There is also an array type in MetaModelica, which is different from the Modelica array type. A MetaModelica array may be updated by a function other than the function it was created in. It is mainly used to store side effects, for example caching results of function calls or efficient hash tables.

3.2.5 Union Types

The `uniontype` declaration in MetaModelica is used to introduce union types. This is similar to the `datatype` concept in the ML family of languages [18]. Consider for example a `Number` type, which can be used to represent several kinds of number types such as integers, rational numbers, real, and complex within the same type.

```
uniontype Number
  record INT
    Integer int;
  end INT;
  record RATIONAL
    Integer dividend, divisor;
  end RATIONAL;
  record REAL
    Real real;
  end REAL;
  record COMPLEX
    Real re,im;
  end COMPLEX;
end Number;
```

The most frequent use of the union type is representation of trees or directed acyclic graphs. A tree is a recursive data type, and representing these is as simple as

using the name of the union type as the type of a field in a record that is part of the union type. There are no restrictions or special syntax required to defined mutually dependent union types as shown by `Expression.VAR` and `Subscript.SUBSCRIPT`.

```

uniontype Expression
  record RCONST "A real constant"
    Real r;
  end RCONST;
  record ADD "lhs + rhs"
    Expression lhs, rhs;
  end ADD;
  record SUB "lhs - rhs"
    Expression lhs, rhs;
  end SUB;
  record MUL "lhs * rhs"
    Expression lhs, rhs;
  end MUL;
  record DIV "lhs / rhs"
    Expression lhs, rhs;
  end DIV;
  record VAR "name[sub1, ..., subn]"
    String name;
    List<Subscript> subscripts;
  end Var;
end Expression;

uniontype Subscript
  record NOSUB end NOSUB;
  record SUBSCRIPT
    Expression subscript;
  end SUBSCRIPT;
end Subscript;

```

4 Implementation

This section provides the details about the implementation of the compiler and the runtime system for the MetaModelica extensions.

4.1 Compiler

The OpenModelica compiler is implemented mostly in MetaModelica. The front-end, which is where most of OpenModelica development is focused, takes up nearly half of the source code. It elaborates Modelica and MetaModelica code, which is passed on to the backend (for simulations) or directly to code generation (for functions).

The numbers in Table 1 were generated by loading the appropriate sources in the compiler and using the `list()` command to pretty-print the sources. This removes C-style comments, but includes the Modelica-style string comments. The OpenModelica text generation template language Susan [14] is used in the compiler to generate the code generation module.

Table 1. Sizes of OMC Compiler Phases, Lines of Code.

Compiler Phase	Lines
BackEnd (from flat Modelica to sorted eq.syst.)	29190
Code generation (generated code)	35971
Code generation (template source code)	8957
FrontEnd (up to flat Modelica)	92192
OpenModelica scripting environment	21883
Template language Susan compiler	12119
Unparsing modules (pretty-printing data structures)	16984
Utility modules	12983
<i>Total size (excl. generated code)</i>	<i>194218</i>

OpenModelica also requires a runtime system. It is divided into five parts:

- The basic runtime system contains all Modelica builtin function definitions, such as `String()` or `div()` and handles array operations. This runtime needs to be linked to the compiler itself and any source code it produces.
- The simulation runtime system contains the solvers and does event handling.
- The compiler runtime system handles runtime options, settings, CORBA communication and system calls.
- The parser sources are generated by ANTLRv3 [27]. They produce a MetaModelica abstract syntax tree that the compiler can use without further transformations.
- The builtin environment is a Modelica file that contains a list of all builtin functions in the Modelica language as well as the scripting functions that are available in OpenModelica.

Since MMC and OMC external C functions are named and defined differently, they call the same runtime base functions which perform all the work. The lines of code listed in Table 2 exclude the MMC wrapper functions. Header files are included in these measurements, but comments and blank lines are not counted in either headers or source code.

Table 2. Sizes of OMC Parser and Runtime.

Supporting functionality	Lines
Parser (ANTLR sources)	1968
Parser (generated C sources)	49256
Parser (wrapper code)	339
Compiler runtime	6159
Simulation runtime	3405
Basic runtime (excl. MetaModelica and External)	9675
Basic runtime (External code that is maintained in other projects; e.g. Fortran code from LAPACK)	13268
Basic runtime (MetaModelica)	1930
Modelica builtin environment	744
MetaModelica builtin environment	825
Total size (excl. generated and external code)	25045

4.2 Platform Availability

The OpenModelica compiler runs on all the major platforms, including Windows, Mac, and a number of Linux variants. It also runs under .net using its C# code generator. A code generator emitting Java code is under development.

4.3 Language Feature Implementations

4.3.1 Pattern Matching Implementation

There have been three main attempts to add this language feature to the MetaModelica compiler. These attempts all used a regular C stack and exception handling to model `matchcontinue`. In MMC `matchcontinue` is implemented using the continuation passing style [22], which essentially makes exception handling free, at the cost of not having a regular C stack. The main reason for using a regular stack is debugging, which then becomes much easier.

The first approach [34] introduced C-like constructs to the intermediate representation so that it could be mapped to the runtime at an early stage. It created a DFA (Definite Finite Automaton, a deterministic state machine) so that results of pattern-matching in earlier cases could be remembered when matching later cases. The implementation suffered on several points.

First, it worked directly on the abstract syntax, creating a new expression to be elaborated (basically converting the `match`-expression into an internal representation of Modelica that contained `goto`, `label`, `try`, `throw` and `catch` instead of `match`-expressions).

It is hard to change such an implementation or to optimize the code further.

Second, its semantics did not cover the exception handling aspects of `matchcontinue`, so the whole OpenModelica compiler could not be handled by it.

Third, the ways in which it could be used was very specific (only as an assignment statement with variable names as input and output; it could not be nested). The third limitation is minor, since the MMC implementation of MetaModelica did the same thing.

The second attempt to implement `match` expressions was based on the first one, but removing the DFA part. This made the semantics of `match/matchcontinue` work more as expected. However, while `match` expressions could now be nested, it still required variable names as input/output and worked directly on abstract syntax.

Maintaining this code and adding special cases for language constructs proved futile. Again, it was not possible to completely cover the semantics of `match`-expressions.

For example, it is impossible to translate the type of the empty list back to abstract syntax (you cannot write `list<Any>` in a program, even though the type system uses this type for the empty list internally).

The start of the final attempt was the implementation of pattern-matching assignments, e.g.:

```
(a,1.0) := fnCall(...)
```

Previously, this was handled by converting the statement to nested `matchcontinue` blocks, which did not always work. This implementation of pattern-matching was then used to express `match` expressions, which are now treated as expressions instead of statements.

They have result types like any other expression and may even be the input of another `match` expression `match`, e.g.:

```
match (match str
  case "Modelica" then true; else false;
end match)
  case true then 3.0; else 2.0;
end match;
```

This approach is much simpler to maintain because it works on the correct level of abstraction. It has access to full type information while doing the translation.

While it does work correctly for all cases, it is a bit slower than the first implementation. It no longer has a DFA to avoid pattern-matching the same thing several times. However, because the code for pattern-matching is essentially created during code-generation instead of during elaboration, it enables us to switch from a C++ runtime to something else if C/C++ is not powerful enough. It also enables us to do optimizations based on patterns. For example, we can detect dead code (un-

reachable patterns) or detect if we can select a case directly instead of doing a linear search of all patterns.

4.3.2 Union Types Implementation

To add support for the `Array`, `Option`, `List` and `Tuple` types to a Modelica compiler is not an easy task. Array expressions sometimes need to be type converted to lists, but there are more expressions than the array data constructor that elaborate to the same expression. For example, the `fill(3,2)` operator call produces an array expression `{3,3}`. This expression can be converted to a list.

The `Option` type was easier to implement than the other types introduced in MetaModelica. The reason for this is that its syntax does not overlap with arrays (like lists), or multiple outputs of functions (like tuples). Tuples are treated differently in the runtime depending on if they are multiple outputs of a function or a tuple.

The union type syntax looks like it would be accessed by `Package.Uniontype.RECORD`, but in MMC the record is implicitly added to the package scope so it is accessed as `Package.RECORD`. In order to bootstrap the compiler, the same principle is used in OpenModelica. The type of a call to a record constructor is the union type, not the record type itself, which means accessing fields is limited to pattern-matching.

4.3.3 Polymorphism

There could be several different ways to implement polymorphic functions in a Modelica compiler. We use Hindley-Milner-style type inference [15][17] to infer the type of a function based on its interface.

That is, we only consider inputs and outputs of functions when doing type inference. From the inputs of a call expression, we create constraints that we unify. If we succeed, we have determined the actual type of each type variable, which is used to calculate the result type of the call. All other variables have a type and no type inference is done for local variables.

Note that neither Modelica nor MetaModelica has the concept of a `Number` type (some Modelica builtin operators take either `Integer` or `Real` as input, but this cannot be represented in Modelica code). Because there is no `Number` type, a call to `valueEq(1,1.5)` would not pass type checking because the types are different.

While implementing polymorphic functions in a Modelica compiler is rather straight forward, there are a lot of things to think about when solving the type constraints. The names of type variables can be from the current scope, from the called function or from a function pointer used as an input argument. Keeping track

of the different sources is the key to getting the correct semantics.

4.4 Runtime System

Because the OpenModelica compiler runtime only handled the static Modelica structures we had to extend the runtime system to handle the new functional constructs.

4.4.1 Data layout

The same layout of values (objects) in memory as for the MMC compiler runtime was used. Basically all values besides integers are boxes that contain a header followed by the actual data. The pointers are tagged (the small number 3 is added to them) to be able to differentiate between pointers and other values such as integers. The differentiation is needed for garbage collection. The headers have 32 bits or 64 bits depending on the platform (32/64 bit platforms). Inside the header the bits are split into:

- slots – represents the size of data in words
- constructor – represents the type of the data (i.e. index in an uniontype or string, real, etc)

On 32 bit platforms the slots are 22 bits and constructor (the tag) is 10 bits. On 64 bit platforms the slots are 54 bits and 10 bits are devoted to the constructor (the tag).

Integers are either 31 or 63 bits depending on platforms and are represented as even values (i.e. integer N is represented as $N \ll 1$, N shifted left by 1).

Bit zero (0) is zero if the box (node) contains pointers and 1 otherwise.

A list value is represented as a box containing a `CONS` header, a pointer to the element and a pointer to next. The end of the list is represented by a box containing a `NIL` header with zero slots.

An `Option` value is represented as a box containing a `SOME` header and a pointer to the element or a `NONE` header with zero slots.

4.4.2 Builtin MetaModelica functions

New built-in functions are needed to perform operations on the MetaModelica boxed values. The functions are can be used directly in the MetaModelica code or they are generated in the C code from operators. We can classify these functions based on the types they operate on.

- **Booleans:** `boolEq`, `boolString`
- **Integers:** `intEq`, `intLt`, `intLte`, `intGt`, `intGte`, `intNe`, `intString`, `intAdd`, `intSub`, `intMul`, `intDiv`, `intMod`, `intMin`, `intMax`
- **Reals:** `realEq`, `realLt`, `realLte`, `realGt`, `realGte`, `realNe`, `realString`, `realAdd`, `real-`

Sub, realMul, realDiv, realMod, realMin, realMax

- **Strings:** stringEq, stringCompare, stringHash, stringGet, stringAppend
- **Lists:** listAppend, listMember, listGet, listReverse, cons (:: operator), stringAppend-List
- **MetaArrays:** arrayCreate, arrayGet ([x] index operator), arrayCopy, arrayUpdate

4.4.3 Garbage collection

Garbage collection (GC), i.e., automatic memory reclamation, of un-used heap-allocated data is a must for a functional language to be able to collect unused values and reuse memory. The garbage collector used in MMC is a simple 2-generational copying compacting garbage collector [28][36]. It has two main memory areas: a young generation where the objects were initially allocated and an older region to which the objects are promoted if they survive a minor collection. We could not reuse this old MMC garbage collector for the OMC runtime because *it moves pointers* and makes it hard to write external functions.

A new garbage collector (GC) has been implemented in OMC. It is a mark-and-not-sweep GC which *does not move* pointers to new addresses. Instead, it marks the unused values (objects) and these can be used for the new allocations. This kind of collector has a better memory usage than the old MMC GC which could only use half of available memory (the other half was needed for the reserve region).

Currently GC can only happen immediately after a function is entered. Basically, the GC starts from a list of roots and marks all values that are reachable from these roots (transitive closure). Then all regions that are not marked become part of a free list (and is garbage) that can be reused for new allocations.

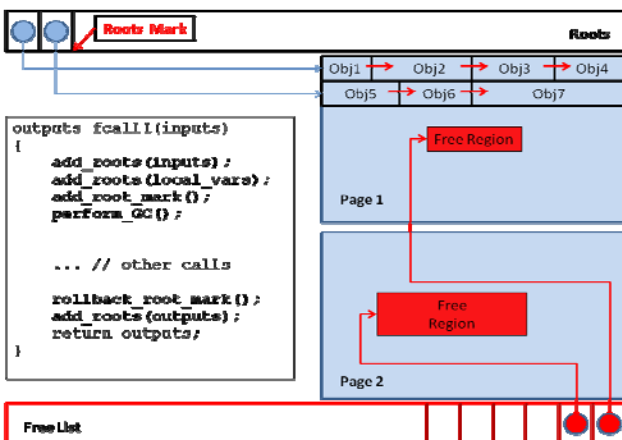


Figure 2. Roots and memory regions during garbage collection.

When we enter a function we add all inputs to the root list and we mark the position in this list. If the function fails we rollback the roots mark to the previous mark and everything allocated above that mark becomes garbage. If the function succeeds the roots mark is rollback and the outputs are added to the roots.

As a future improvement we can allow local allocations in functions as part of roots and then GC could run at any allocation. This way loops that allocate data can perform GC and not exhaust memory. Also we could remove from roots the pointers allocated in the loops because when a loop is finished these pointers should be reachable from variables in the enclosing scope.

4.5 Issues

We identified problems with the tools we use, MMC and OMC. However, there are also design issues in the Modelica language that we needed to work around in order to bootstrap OMC.

4.5.1 MMC Problems

The old MMC compiler has several problems. While it has very good performance, maintaining the code is cumbersome. We would like to have only one tool to maintain, the OpenModelica Compiler. Still, before we can switch to using the OpenModelica Compiler as the default MetaModelica Compiler, we need to be able to compile the same code as MMC during a transition period. The problem is that the MetaModelica to RML translator is very relaxed when it comes to what syntax it accepts.

In order to find some common errors, we traverse the whole abstract syntax and verify that all `matchcontinue` expressions have the same input and output as the function has. This is a limitation of MMC, but it does not actually check that this assertion holds. Moreover, while MMC does some type checking, it does not do it for expressions of the type:

```
x = fnCall(x);
```

For these expressions, the lhs and rhs `x` may have different types and MMC may allow some code that is not valid MetaModelica code. Finding such code and rewriting it takes a lot of time, but does produce code that is easier to maintain.

Other issues include allowing code like:

```
import Env;
type Env = Env.Env;
```

In the Modelica language the import will essentially be ignored and the tool will probably get a stack overflow because the type `Env` is recursive on itself.

Code that looked like this had to be refactored.

4.5.2 OpenModelica Issues

Some parts of the compiler were rewritten so that it was easier to detect common errors that MMC does not handle. As a result, the OpenModelica Compiler now has vastly better error messages for algorithmic code. We have also started propagating file, line and column information to more error messages because when you have a large application that you want to fix, it is essential that you find the correct line quickly. Improving the error messages probably saved more time than it cost to implement and as a result, the compiler should be giving better error messages both for Modelica and Meta-Modelica code.

OpenModelica has issues with shadowing certain builtin operators. For example, creating a function `ndims` and calling it in the same package will result in the Modelica builtin `ndims` operator being used instead of this function. A few of these functions existed in the compiler and had to be renamed (rewriting `Lookup` in OpenModelica is an alternative, but would take more time).

4.5.3 Modelica Issues

The Modelica Specification [17] says that external functions map `Integer` to `int`, with no way to change their behavior. If external functions in the compiler runtime need to access large integers, this is a problem. Examples of such a function are `referenceEqual`, which checks if two pointers are equal and the `stringHash` family of functions. Values were being truncated because of the limited precision of `int` on 64-bit platforms, so we had to move functions from external C functions in the compiler into the MetaModelica language itself.

5 Performance

While we are not fully satisfied with the performance of the compiler at the time of writing this text, OMC compiled by OpenModelica has only been running for a few weeks. This initial version lacks a garbage collector; we simply allocate memory until we run out. Even without memory management, the majority of the OpenModelica regression and unit tests run.

The performance of the working examples varied widely and some tests could be heavily improved by doing small changes.

In the first executable version (a few weeks ago) of the bootstrapped compiler even simple test cases were nine times slower than in the MMC version. Since the new version of OMC uses a real C stack, it is possible

to use general-purpose debugging and profiling tools, such as `gdb`, `gprof` or `valgrind` on it.

Using `valgrind --tool=callgrind`, we could see that the `PEXPipe` example spent around 80% of its time doing exception handling. The C++ try-throw-catch feature had been used to implement the exception handling [31] of `matchcontinue` expressions. C++ exceptions are slow because they are not supposed to be used often. We rewrote the exception handling using `setjmp/longjmp`. Since we do not use C++ classes or C++-heap-allocated data, this level of exception handling is sufficient for us.

The new exception handling resulted in a 10x speed-up for certain examples (on the average 3x). Exception handling still uses approximately ~25% of the compiler execution time. We also implemented optimizations that rewrite `matchcontinue` to `match` automatically as a compiler optimization. This reduced some of the `setjmp` overhead.

5.1 Benchmarks

All benchmarks were performed using a laptop running 64-bit Ubuntu 10.10. The machine was equipped with a dual-core Intel Core i7 M620 (2.67GHz) and 8GB of memory. All file IO was performed on RAM-disk in order to reduce disk overhead. Both compilers were compiled using GCC 4.4.5 with optimization level set to `-O3`.

The two implementations use virtually the same parser, so speed here is identical. The Modelica Standard Library (MSL) is distributed in multiple files that are parsed and merged into a single tree by the OpenModelica Compiler. The version of MSL used was 3.1. MSL 3.1 is 7.8MB stored as a single file contains 922 models and 615 functions.

Unparsing the tree is the process of going from the internal tree structure back to Modelica concrete syntax (source code). Here MMC suffers from an additional performance penalty because the tree was created in an external C function and not directly on its heap.

Table 3 Compiler performance, parsing

Task	MMC	OMC	Factor
Parse MSL3.1, single file	1.577s	1.577s	1x
Parse MSL3.1, multiple files	1.297s	1.253s	0.97x
Parse MSL3.1+Unparse	3.674s	2.172s	0.59x
Unparse only	2.377s	0.600s	0.25x

Table 4 shows the performance of OMC for simple tests, ~1x the speed of MMC.

Table 4 Compiler performance, simple tests

Task	MMC	OMC	Factor
drmodelica, 104 flattening tests	2.136s	2.049s	0.96x
mofiles, 548 flattening tests	12.770s	13.409s	1.05x
misl 1.5, 60 flattening tests	1.726s	1.578s	0.91x
mosfiles, 120 simulation test	110.67s	107.96s	1.03x

The tests are limited to 5GB of virtual memory through the use of `ulimit`. This rather low limit was chosen because then swap memory would never be accessed. Because of this limit, a few tests fail. The bootstrapped compiler has issues running large models. For example, running `checkModel` command on the `EngineV6Analytic` model, which has 9491 equations, uses roughly 7GB of memory. The overall performance shown in Table 5 is acceptable.

Table 5 Compiler performance, big tests

Task	MMC	OMC	Factor
EngineV6Analytic, <code>checkModel</code> , time	55.908s	55.553s	0.99x
EngineV6Analytic, <code>checkModel</code> , memory	400MB	7GB	17.5x
All 1379 tests, single thread, time	24m8s	20m55s	-
All 1379 tests, number of failed tests	13	30	-
Excluding failing tests, 1349 total	17m25s	18m	1.03x

There is a difference in how the two tool chains create the executable compiler. MMC is created by translating each package into a C-file, compiling them separately, and linking everything together. OMC is created by translating all packages into a single 31MB C-file. While this may produce a faster executable, the whole compiler needs to be recompiled even if only one package changes. We are working on supporting both methods in the future, including fine-grained separate compilation for faster turn-around.

As Table 6 shows, compiling OMC using OMC is significantly faster than using MMC. The size of a non-debug executable was calculated by stripping it of all debug symbols and tracing functionality. Since the bootstrapped compiler compiles quite fast, we can spend more time on optimization algorithms. MMC uses code instrumentation when compiling a debug executable. The bootstrapped compiler has a C stack

and simply adding debug symbols is sufficient to run a debugger or profiler on it.

Table 6 Compilation speed

Task	MMC	OMC	Factor
Translate sources to C	98.8s	49.7s	0.50x
Compile C, <code>gcc -O3</code>	233.3s	152.0s	0.65x
Executable size	18.7MB	5.43MB	0.29x
Compile C, <code>gcc -Os</code>	218.9s	102.8s	0.47x
Executable size	10.3MB	5.10MB	0.50x
Debug version, <code>gcc -g -O3</code>	1405s	152.5s	0.11x
Executable size	125MB	25MB	0.20x

6 Related Work

Functional programming languages often bootstrap themselves during the build process. Some also include integrated lexer and parser generators specific to their own programming language, which is something MetaModelica is currently missing, but under development.

In the Lisp family [35] bootstrapped compilers have been available since the 1960's. These languages are strongly typed, like MetaModelica, but defer the check to runtime instead of compile-time. This often leads to error messages that are hard to understand because you don't always know where a piece of data originated if you get a type error.

Objective Caml [25], SML/NJ [1] and MLton [19] are examples of bootstrapped compilers in the ML family. These languages are similar to MetaModelica in that they both use very similar language constructs and type inference. This should come as no surprise since RML/MMC is written in Standard ML and compiles using either SML/NJ or MLton. One major difference is that all variables in MetaModelica have a specific type while in ML each expression has a most general type. MetaModelica can generate error messages that are easier to understand because type inference only has to be performed when calling a polymorphic function. However, it also results in more local variable declarations since all temporary variables need to be declared. This is both positive (you document what type you expect a variable should have) and negative (you end up with a lot of local variables).

The concept of the `matchcontinue` expression is something that the ML family is missing. It is instead possible to use explicit exception handling or use guard expressions to prevent a case from actually matching a pattern, which is often sufficient. The ML family of languages also has lambda functions, which is currently missing in MetaModelica.

The `matchcontinue` expression is more general than the `match` expression, which is common in functional programming. It is related to clauses in logic programming since it provides backtracking on failure. However, in Prolog [24] there are usually many possible answers to a given logic program since it evaluates combinations of clauses that satisfy the program. In MetaModelica only the first valid answer is returned and no subsequent case is evaluated. Thus, MetaModelica is more efficient and consistently statically strongly typed, whereas logic programs sometimes can be expressed more concisely.

7 Conclusions

We have demonstrated a Modelica compiler that is capable of compiling itself. Thus, it is possible to describe all of the semantics of Modelica in a slightly extended version of Modelica. We have also shown that the performance of the implementation is sufficient. The effort to achieve these goals was higher than initially expected, and included not only developing the compiler but also a development environment including an Eclipse plugin and a debugger.

Many of the MetaModelica language extensions that allow language modeling are in line with the design goals for Modelica 4 to allow modeling of language features in libraries. We believe that this work will be an important input and proof-of-concept to the Modelica 4 design effort.

7.1 Future Work

The garbage collection implementation needs to be finalized and tested before we can switch to using the bootstrapped compiler for development work.

Once that is done, parts of the compiler can be rewritten/refactored using certain more powerful and concise language constructs in MetaModelica 2.0 [12].

Furthermore, to realize important design goals, a number of language features should be moved into libraries and an enhanced API for accessing compiler functionality from such libraries need to be developed.

The debugger [32] in the Eclipse environment, including a data inspector for the MetaModelica types, needs to be ported to the bootstrapped OMC. Furthermore, the generated code needs to be annotated with line numbers to support the debugger and performance analyzer in mapping to Modelica source file positions.

8 Acknowledgements

This work has been supported by Vinnova in the ITEA2 OPENPROD project, and by SSF in the Provik-

ing HIPo project. The Open Source Modelica Consortium supports the OpenModelica work.

References

- [1] Matthias Blume. The SML/NJ Bootstrap Compiler User Manual. July, 2001.
- [2] David Broman and Peter Fritzson. Higher-Order Acausal Models. In *Proceedings of the 2nd International Workshop on Equation-Based Object-Oriented Languages and Tools, (EOOLT'2008)*, Pathos, Cyprus, July 8, 2008. Published by Linköping University Electronic Press, <http://www.ep.liu.se/ecp/024/>, July 2008.
- [3] David Broman. *Meta-Languages and Semantics for Equation-Based Modeling and Simulation*. Dissertation No 1333, www.ep.liu.se, Linköping University, October 1, 2010.
- [4] Stefan Brus. Bootstrapping The OpenModelica Compiler: Implementing Functions As Arguments. Master thesis draft, 2009. www.ep.liu.se. Finalized Spring 2010.
- [5] Emil Carlsson. Translating Natural Semantics to Meta-Modelica. Master Thesis, LITH-IDA-Ex--05/073—SE, Linköping University, October 2005.
- [6] Peter Fritzson. *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*, 940 pages, Wiley-IEEE Press, 2004.
- [7] Peter Fritzson, Peter Aronsson, Håkan Lundvall, Kaj Nyström, Adrian Pop, Levon Saldamli, and David Broman. The OpenModelica Modeling, Simulation, and Software Development Environment. *Simulation News Europe*, 44/45, Dec. 2005. <http://www.openmodelica.org>
- [8] Peter Fritzson, Adrian Pop, and Peter Aronsson. Towards Comprehensive Meta-Modeling and Meta-Programming Capabilities in Modelica. In *Proc. of the 4th International Modelica Conference*, Hamburg, Germany, March 7-8, 2005.
- [9] Peter Fritzson. *Language Modeling and Symbolic Transformations with Meta-Modelica*. (Later versions [10], [11]), www.ida.liu.se/~pelab/Modelica, and www.openmodelica.org Version 0.5, June 2005.
- [10] Peter Fritzson. *Modelica Meta-Programming and Symbolic Transformations - MetaModelica Programming Guide*. (Slightly updated version of [9], later update: [11]) <http://www.openmodelica.org/index.php/developer/devdocumentation>. June 2007.
- [11] Peter Fritzson and Adrian Pop. *Meta-Programming and Language Modeling with MetaModelica 1.0*. (Almost identical to [10], but

- tech. report). Technical reports in Computer and Information Science, No 9, Linköping University Electronic Press, <http://www.ep.liu.se/PubList/Default.aspx?SeriesID=2550>, January 2011.
- [12] Peter Fritzson, Adrian Pop, and Martin Sjölund. *Towards Modelica 4 Meta-Programming and Language Modeling with MetaModelica 2.0*. Technical reports in Computer and Information Science, No 10, Linköping University Electronic Press, <http://www.ep.liu.se/PubList/Default.aspx?SeriesID=2550>, February 2011.
- [13] Peter Fritzson, Adrian Pop, David Broman, Peter Aronsson. Formal Semantics Based Translator Generation and Tool Development in Practice. In *Proceedings of ASWEC 2009 Australian Software Engineering Conference*, Gold Coast, Australia, 2009.
- [14] Peter Fritzson, Pavol Privitzer, Martin Sjölund, and Adrian Pop. Towards a text generation template language for Modelica. In *Proceedings of the 7th International Modelica Conference*, Como, Italy, Sept. 20-22, 2009
- [15] J. Roger Hindley. The principal type scheme of an object in combinatory logic. *Transactions of the American Mathematical Society*, 146:29-60, Dec. 1969.
- [16] Kenneth C. Louden. *Programming Languages, Principles and Practice*. ISBN 0-534-95341-7, Thomson Brooks/Cole, 2003.
- [17] Robin Milner. A Theory of Type Polymorphism in Programming. *Journal of Computer and System Sciences*, 17:248-375, 1978.
- [18] Robin Milner, Mads Tofte, Robert Harper and David MacQueen. *The Definition of Standard ML*, 128 pages, MIT Press, 1997.
- [19] MLton. Installation Instructions. Jan. 2011. <http://mlton.org/PortingMLton>.
- [20] Modelica Association. *The Modelica Language Specification Version 3.2*, March 2010. <http://www.modelica.org>.
- [21] Modelica Association. *Modelica Standard Library 3.1*. Aug. 2009. <http://www.modelica.org>.
- [22] Modelica Association. Minutes of the Modelica Design Meeting 67, www.modelica.org, Atlanta, Georgia, September 2010.
- [23] Henrik Nilsson, John Peterson, and Paul Hudak. Functional Hybrid Modeling from an Object-Oriented Perspective. In *Proc. of EOOLT'2007*, LIU Electronic Press, www.ep.liu.se, Berlin, Germany, August, 2007.
- [24] Ulf Nilsson and Jan Maluszynski. *Logic, Programming and Prolog*. Wiley, 1995.
- [25] Objective Caml 3.12.0 installation notes. Aug, 2010. <http://caml.inria.fr>
- [26] Martin Odersky, Lex Spoon, and Bill Venner. *Programming in Scala*. Artima Press, 2008.
- [27] Terence Parr. ANTLR Parser Generator 3.3, Accessed November 2010. <http://antlr.org/>
- [28] Mikael Pettersson. *Compiling Natural Semantics*, Department of Computer and Information Science, Linköping University, PhD Thesis No. 413, 1995. Published in *Lecture Notes in Computer Science* No 1549, Springer Verlag, 1999.
- [29] Adrian Pop, Peter Fritzson, Andreas Remar, Elmira Jagudin, and David Akhvlediani. OpenModelica Development Environment with Eclipse Integration for Browsing, Modeling, and Debugging. In *Proc 5th International Modelica Conf. (Modelica'2006)*, Vienna, Austria, Sept. 4-5, 2006.
- [30] Adrian Pop and Peter Fritzson. MetaModelica: A Unified Equation-Based Semantical and Mathematical Modeling Language. In *Proceedings of Joint Modular Languages Conference 2006 (JMLC2006)* Published in Lecture Notes in Computer Science No 4228, ISSN 0302-9743, Springer Verlag. Jesus College, Oxford, England, Sept 13-15, 2006.
- [31] Adrian Pop, Kristian Stavåker, and Peter Fritzson. Exception Handling for Modelica. In *Proceedings of the 6th International Modelica Conference (Modelica'2008)*, Bielefeld, Germany, March.3-4, 2008.
- [32] Adrian Pop. *Integrated Model-Driven Development Environments for Equation-Based Object-Oriented Languages*. www.ep.liu.se, PhD Thesis No. 1183, Linköping University, June 5, 2008.
- [33] Martin Sjölund. Bidirectional External Function Interface Between Modelica/MetaModelica and Java. Master thesis, IDA/LITHEXA09/041SE, Aug 2009.
- [34] Kristian Stavåker, Adrian Pop, and Peter Fritzson. Compiling and Using Pattern Matching in Modelica. In *Proceedings of the 6th International Modelica Conference (Modelica'2008)*, Bielefeld, Germany, March.3-4, 2008.
- [35] Guy L. Steele Jr. and Richard P. Gabriel. The Evolution of Lisp. In *Proceedings of the Conference on History of Programming Languages*. New York, April, 1993.
- [36] Paul R. Wilson, Uniprocessor garbage collection techniques, Lecture Notes in Computer Science, Volume 637/1992, page 1-42. Springer Verlag. 1992.
- [37] Dirk Zimmer. *Equation-Based Modeling of Variable Structure Systems*. PhD Dissertation, ETH Zürich, 219 pages, 2010.

A Scade Suite to Modelica Interface

Daniel Schlabe, DLR Institute of Robotics and Mechatronics - Daniel.Schlabe@dlr.de
 Tobias Knostmann, Esterel Technologies GmbH - Tobias.Knostmann@esterel-technologies.com
 Tilman Bunte, DLR Institute of Robotics and Mechatronics - Tilman.Buente@dlr.de

Abstract

This article presents implementation and utilization details of the currently developed interface from Scade Suite to Modelica. By a few clicks one can generate a Modelica block from Scade Suite models that can be directly used and simulated in Modelica. This block calls an external function periodically, where the C-code generated by Scade Suite is invoked.

The main purpose of the interface is to test the generated C-code within a simulated environment which is also known as Software in the Loop (SIL).

Keywords: Scade Suite; Modelica interface; C-code integration; Software In the Loop

1 Introduction

1.1 Scade Suite Description

The acronym Scade stands for Safety-Critical Application Design Environment. The term describes a Suite of model-based software development and verification tools as well as the modelling language itself. This language is formally defined and proven to be fully deterministic, hence it allows certified and qualified code generation for safety-critical systems. The textual base of the language is an extension [3] of the synchronous dataflow language Lustre [10]. The textual language is by default hidden behind a design environment for graphical models featuring deterministic state machines, dataflow block diagrams and decision diagrams.

Most importantly, this formal technology allows to create a seamless process to leverage a large degree of automation:

- model editing
- verification by means of simulation and formal methods
- traceability
- documentation generation

- code generation

It also enables circumvention of cumbersome activities demanded by today's critical software standards such as DO-178B, EN50128 or IEC61508 (see [5] and [6]).

The qualification and certification of the Scade KCG code generator demands that the produced code is not altered in any way. Therefore, in order to adapt the code regarding specific calling conventions of e.g. Modelica functions, the developer needs to keep the KCG code unchanged and add C-code to meet the interface requirements.

To automate the generation of this glue code, Scade offers a Tcl-based [12] scripting environment that runs in parallel to the code generation process. It offers access via a specific application programming interface (API) to the model structure information and the translation patterns of Scade-language based objects to C constructs. This method is used to provide adaptors to various real-time operating systems and other code-wrapping targets, all of which can be adapted to the users needs. We will describe how we use this means to create our Modelica adaptor.

1.2 Scope of the interface

Thinking of a Scade Modelica interface one can imagine two possible interface "directions". The first one is to integrate Code generated by Scade Suite in Modelica. The second one is to generate a Scade Suite model out of a Modelica model. The herein developed interface deals with the first interface direction. Certainly, the second direction is very interesting since the generated Scade code would be certified. But this will still need a lot of investigations and work. Furthermore it would be restricted to a subset of Modelica and not the whole language. See chapter 4 for an outlook of this point.

By using the herein developed Scade-Modelica interface it is rather possible to test the generated C-code at a very early stage of the design process

by means of simulated environments, also known as Software in the Loop (SIL). This will reduce development time since it is not always necessary to implement the code on the target system. Furthermore it is possible to specify requirements beforehand by means of Modelica model environments that contain simple placeholders for the functions to be developed. This will reduce iteration loops with the client who specified the requirements and enables therefore a fast software development.

The integration of any C-code into Modelica could be done manually indeed. However, this requires creating and modifying interface files like C-code or Modelica code every time the model is changed, which is quite inconvenient. Using the developed automatic interface one can easily integrate new C-code into Modelica and test it in the simulated environment.

1.3 Application Areas

Typical application areas will be the aerospace and automotive industry or any other field where a safety critical function should be integrated into a complex system. Any developed function or controller can be tested in a simulated environment or object using the interface. A Modelica block can easily be generated as long as the Scade model fulfils the restrictions described in section 2.1. For instance, energy management for more/all electric aircraft or hybrid/electric vehicles is a topic of rising interest that becomes more and more complex, where interactions with the system model as well as with other management functions or controllers can often not be neglected. Therefore it is useful to be able to simulate the management function together with the aircraft or vehicle model.

2 Implementation

2.1 Basic principle and restrictions

The basic principle of the developed interface is to integrate C-code into Modelica through external functions. Therefore, an adaptor for code integration to Modelica has been developed for Scade Suite. So the KCG-code can be used without any changes. The adaptor generates some additional interface files and a Modelica package. This folder contains:

- Unchanged KCG-code
- Additional C-code for the interface function and for Scade state vector (see next sections for more

details)

- An image
- Modelica package containing one package.mo file

The additional code, the folders and the .mo files are generated via Tcl scripts. These scripts can be easily executed during code generation in Scade Suite.

For the current version of the interface the following restriction applies: only Integer, Boolean and Real are allowed as input and output signal types. That means that for each input a separate connector will be displayed in Modelica. On the one hand, this eases readability. But on the other hand, with more complex models having many inputs and outputs, the Scade block will be very large. A vector of Boolean, Integer and Real could technically be possible, but this would seriously affect readability since the variable would be identified with an index instead of its original name. For this reason, this method was not implemented in the current version. Also, records are not supported yet. In fact, Dymola doesn't support records in external C-code correctly.

Provided that the input and output signals are of the supported types as described previously, there is no further restriction regarding complexity or internal states of the Scade Model. The interface will also work for large numbers of inputs and outputs, though the resulting block illustrated in Modelica will be very large as well.

2.2 Implementation Details

The implementation details will be illustrated by way of an example. Figure 1 displays the Scade model of a

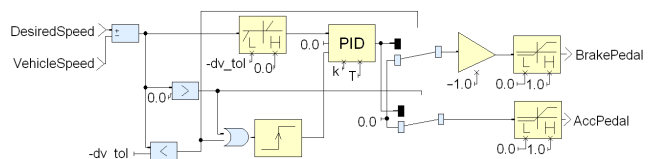


Figure 1: Speed Controller in Scade Suite

vehicle speed controller that outputs brake and acceleration pedal positions using the current speed of the vehicle and the desired speed as an input. The Modelica adaptor for Scade Suite will generate an additional tab. As shown in figure 2 just three items are required:

- The target directory,
- Modelica project name = Modelica package name,

- The Modelica version to be used.

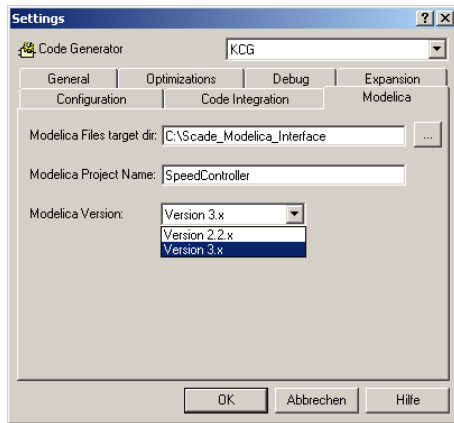


Figure 2: Modelica Tab in Scade

The interface is tested for Modelica versions 2.2.2, 3.0 and 3.1. There were some changes in the annotation syntax between Modelica 2.2.2 and 3.0, which the interface takes into account. So if a wrong Modelica version is chosen, important graphic elements like the connectors will not be displayed.

The respective C-files and a Modelica Package are now generated containing the following elements:

- block ScadeBlock
- class ScadeStateVector
- function ScadeStep

The basic element is the function ScadeStep, where the C-code is integrated via external functions:

```
function ScadeStep
  input Real vehicleSpeed ;
  input Real desiredSpeed ;
  input ScadeStateVector ssv;
  output Real brakePedal ;
  output Real accPedal ;
  external "C" ScadeStep(vehicleSpeed,
desiredSpeed, ssv, brakePedal, accPedal);
  annotation(Include="
    #include <../SpeedControl.c>
    #include <../PID_BaseClass.c>
    ...
",
  uses(Modelica(version="3.0")));
end ScadeStep;
```

The C-code is included using absolute paths. So the working directory doesn't need to be the package

directory. This implies that one has to modify the absolute paths here manually if the package is moved or copied elsewhere.

The inputs and outputs of the Scade model are mapped one-by-one to the Modelica function. If the Scade model has any dynamic states one extra input is needed for the function: the so called Scade state vector. Since functions in Modelica are not allowed to have any internal state, the previous state will be an input for each call of the function. Going back to the speed control example, the PID controller has some internal states. The corresponding Scade state vector is specified in Modelica as follows:

```
protected class ScadeStateVector
  "External Scade States"
  extends ExternalObject;
  function constructor
    output ScadeStateVector ssv;
    external "C" ssv =
initScadeStateVector();
  end constructor;
  function destructor "Release storage of
ScadeStateVector"
    input ScadeStateVector ssv;
    external "C" freeScadeStateVector(ssv);
  end destructor;
end ScadeStateVector;
```

Using external C-code generated by the Modelica adaptor, memory is allocated by the constructor and freed by the destructor. The ScadeStep function and the ScadeStateVector class are declared protected since the user shouldn't use them directly.

The user interface in Modelica is represented by a Scade block (see figure 3) that can be used per drag and drop. It just needs the sample period and a starting time as parameters. In this Block the function



Figure 3: Scade Block in Modelica

ScadeStep will be called periodically as specified. Furthermore the Scade state vector will be initialized herein:

```
block ScadeBlock
  "Scade Suite Block containing standard
  interfaces generated by Scade"
```


viewer *SimVis* provided with the DLR Visualization library.

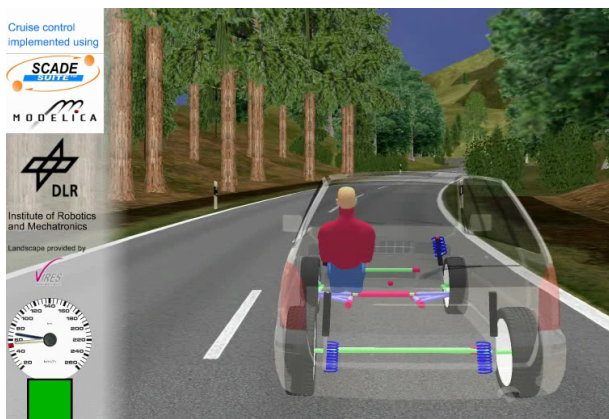


Figure 6: Visualisation in SimVis

4 Conclusion and way forward

A first version of an interface from Scade Suite to Modelica has been presented in this paper. The interface will be adapted to future versions of Modelica and Scade Suite if needed. One planned enhancement is the removal of absolute paths for including C-code. This can be replaced by URIs introduced in Modelica version 3.1. It is also possible to use the Functional Mock-Up Interface (FMI) instead of external functions for future versions.

Another very interesting thought is to generate Scade-Models directly out of Modelica. This would enable the automatic generation of certified code from a Modelica model. Certainly it can be expected that some restrictions will apply to the Modelica model. This opposite interface direction can be a challenging task for future investigations.

5 Acknowledgements

This work has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) for the Clean Sky Joint Technology Initiative under grant agreement n° CSJU-GAN-SGO-2008-001 [8].

References

- [1] Bellmann, Tobias (2009) Interactive Simulations and advanced Visualization with Modelica.
- [2] Bunte, Tilman; Chrisofakis, Emanuel (2011) A Driver Model for Virtual Drivetrain Endurance Testing. In: Proceedings of the 8th International Modelica Conference. Linköping University Electronic Press. Modelica Conference, 20.-22. March 2011, Dresden, Germany.
- [3] Colaço, JeanLouis; Pagano, Bruno; Pouzet, Marc (2005) A Conservative Extension of Synchronous Dataflow with State Machines. In: EM-SOFT'05 Sept. 9-22 2005, Jersey City, New Jersey, USA.
- [4] Dempsey, Mike. An introduction to the VehicleInterfaces package. Tutorial at Modelica conference 2006, Vienna, 2006.
- [5] Fornari, Xavier. Understanding How Scade Suite KCG Generates Safe C Code. 2010. White Paper of Esterel Technologies. [online]: <http://www.esterel-technologies.com/technology/WhitePapers/>
- [6] Pagano, Bruno; Andrieu, Olivier; Moniot, Thomas; Canou, Benjamin; Chailloux, Emmanuel; Wang, Philippe; Manoury, Pascal; Colaço, Jean-Louis. Experience Report: Using Objective Caml to develop safety-critical embedded tools in a certification framework. In: Proceedings of the 14th ACM SIGPLAN international Conference on Functional Programming. Edinburgh, Scotland, August 31 - September 02, 2009
- [7] Tobolar, Jakub; Otter, Martin; Bunte, Tilman. Modelling of Vehicle Powertrains with the Modelica PowerTrain Library. In: Systemanalyse in der Kfz-Antriebstechnik IV, Seiten 204-216. Dynamisches Gesamtsystemverhalten von Fahrzeugantrieben, Augsburg, 2007.
- [8] Clean Sky project homepage [online]: <http://www.cleansky.eu>
- [9] EUROSYS LIB Project Profile 2007 [online]: http://www.itea2.org/public/project_leaflets/EUROSYS LIB_profile_oct-07.pdf
- [10] The synchronous dataflow programming language LUSTRE (1991) [online]:

[http://citeseer.ist.psu.edu/viewdoc/
summary?doi=10.1.1.34.5059](http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.34.5059)

- [11] OpenDRIVE project [online]
<http://www.opendrive.org>
- [12] Tool command language (Tcl). [online]:
www.tcl.tk or [http://citeseer.ist.psu.edu/viewdoc/
summary?doi=10.1.1.38.8230](http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.38.8230)

Towards a model driven Modelica IDE

Roland Samlaus¹ Claudio Hillmann¹ Birgit Demuth² Martin Krebs²
 Fraunhofer Institute for Wind Energy and Energy System Technology¹
 Technische Universität Dresden, Institut für Software- und Multimediatechnik²

Abstract

Model Driven Software Development evolved into a common way of creating software products. Describing software in a more abstract way simplifies and speeds up the development process and generated code turns out to fulfill high quality standards. As a subcategory of model driven development Domain-Specific Languages concede to express problems in a domain specific way. By defining a languages grammar, an editor that provides basic support for developers can be generated automatically. This paper describes how these concepts are utilized for the creation of a Modelica Integrated Development Environment (IDE). Helpful functionality is implemented in a model driven way to maximize assistance during the development process. Thus the developer receives a tool that allows to survey large scale projects and provides functionality that is well known in other popular programming languages. Furthermore an approach for semantical verification of Modelica documents during the development process is presented. This allows to detect and correct errors early.

Keywords: Modelica, IDE, OCL, verification

1 Introduction

In the last years the importance of Modelica in the field of engineering increased significantly. Many companies utilize the language for modeling and simulation of physical systems. Thus the support for developers became a vital issue to enable the survey of extensive projects. One approach is the development of libraries as done by the Modelica community. Libraries provide common functionality that can be easily reused and extended and thereby increase the speed of development and ensure high quality of the resulting models.

An additional approach is the usage of development tools. These support the engineer during the implementation of big physical systems (e.g. Dymola¹ and

OpenModelica²). Although the above mentioned tools turned out to be very helpful, supplementary features are desired to ease the handling of complex models. This topic is well known in the software engineering community as well. Therefore we aim at transferring main features to the Modelica world. The structure of our Modelica Integrated Development Environment (IDE) and how it was created by model driven technologies is explained in section 2.

Besides the goal to facilitate daily work for engineers with Modelica, the quality of the outcome has to be addressed. Regarding this purpose the developer has to be encouraged to create syntactically and semantically correct documents. While the syntax can be checked easily, semantic correctness may be hard to prove because semantic constraints may demand extensive calculations. This issue is addressed in section 3 and our approach of utilizing Object Constraint Language (OCL) for verification of Modelica models is stated.

In section 4 some components of the IDE e.g. the editor and views are presented. Finally a summary and future work will conclude this paper in chapter 5.

2 The MDS D approach

Model Driven Software Development (MDS D) is an approach that eases the development process by providing a higher abstraction level of the software that is being developed. Compared to pure code-based development, the architecture can be described in a more conceptual and structured way. A common way is to design the structure and behavior of software with the help of Unified Modeling Language (UML) diagrams. But graphical representation is not the only popular way of an abstract description. In the MDS D community the usage of Domain-Specific Language (DSL)s became more and more important.

DSLs allow the definition of problems in a domain specific way and therefore provide an easy way to ex-

¹<http://www.dymola.com>

²<http://www.openmodelica.org>

press ideas to the domain experts. This can be done graphically as well as in a textual syntax. The Modelica language can also be seen as a textual DSL, since it allows the developer to describe physical systems.

As Eclipse builds the basis for our IDE it is evident to use Eclipse Modeling Framework (EMF) as platform for our models. All data being processed by EMF are based upon ECore that is more or less aligned on Object Management Group (OMG)s Essential Meta Object Facility (EMOF)³ standard. All generated objects extend a basic interface that allows interoperability between objects of different meta-models. More details about the functionality of Ecore can be found on the EMF website⁴.

The project structure of the IDE has been defined with the help of EMF tools. Several ways [8] of describing an Ecore-based meta model are available, like Java annotations or XML, whereas the usage of a UML-like graphical editor might be the most appealing way for developers. The Modelica DSL was created with Xtext⁵, an Open Source framework for the development of DSLs. It also uses the functionality provided by EMF and therefore enables to easily interact with our data models as described in Section 2.2.

In the next sections the creation of the Modelica DSL is described and the underlying data structure explained. Figure 1 gives an overview of the used tools.

2.1 Metamodeling of Modelica with Xtext

For the textual syntax definition of Modelica we use the tool Xtext, that is part of the Eclipse Modeling Project. Xtext’s syntax for defining language grammars closely resembles the Extended Backus-Naur Form (EBNF) notation. Based on this grammar several components are generated automatically. A Tokenizer splits the given text documents into parts that can be interpreted by a parser. The parser that is generated by the parser generator framework ANTLR [3][10] creates an Abstract Syntax Tree (AST) and a Concrete Syntax Tree (CST) of the given text document. Thereby we get an editable tree-like data structure that can be processed by additional software components. The AST represents the structure of a Modelica document. Additionally the CST keeps all information about the concrete representation inside the document e.g. literals and white spaces. Based on the grammar, syntax highlighting and basic code completion for the generated editor is provided. The parser recognizes

syntactical errors and displays them inside the editor and in a separate problems view. The view contains detailed descriptions of the errors and allows to jump into the erroneous area of the document.

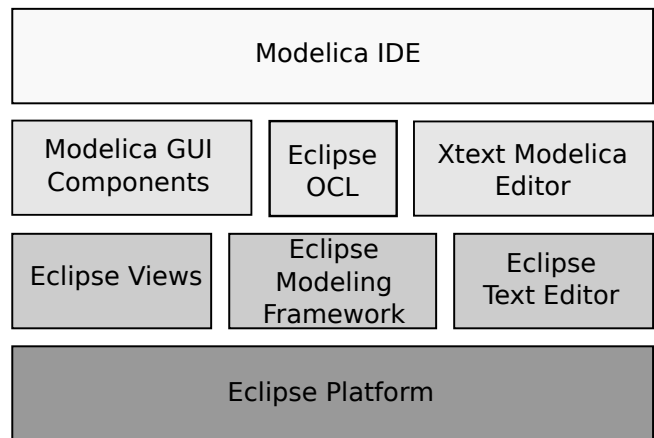


Figure 1: Overview of tools used in the Modelica IDE

Additional to these basic tools, other components are generated that use the AST, for example an outline view that represents the inner structure of opened documents, i.e. inner classes, sub-packages and component declarations. One of the main amenities of Xtext is the integrated resolution of references. Based on this mechanism, the developer can jump to class definitions to investigate implementation details. References are also used as types for component declarations or as extended classes. The references connect ASTs of different documents and thereby form a kind of overlay graph. If referenced objects cannot be found, the affected part of the document is again marked with an error.

However, because of the complex structure of Modelica documents and for performance reasons, special index and linking mechanisms had to be implemented. Based on the index information additional help functions were created, e.g. the proposal of classes that can be referenced at a certain position inside the document.

The usage of Xtext for grammar definitions is covered below by a small example. Listing 1 depicts Modelica’s grammar definition for the element `ImportClause`.

An import clause starts with the keyword ‘import’ followed by an optional abbreviation definition. Optional rules are defined by a question mark. In many Modelica documents `SIunits` are imported with the definition of an abbreviation: ‘SI = Modelica.SIunits;’. This allows to write shorter expressions when using type definitions from the `SIunits` package.

³<http://www.omg.org/>

⁴<http://www.eclipse.org/modeling/emf/>

⁵<http://www.eclipse.org/Xtext/>

```
"import" (abbrev=Name "=")?
reference=[AbstractContent]
wildcard?=".*"? comment=Comment?;
```

Listing 1: Import clause definition in Xtext

The next rule defines a reference to an `AbstractContent`. `AbstractContent` is another grammar rule that represents different kinds of Modelica class contents, e.g. the standard class structure or an extends clause. For more details on the Modelica class structure see [9]. In Xtext square brackets define references to other objects. This is a Xtext specific feature that is not defined in EBNF. A generic linking mechanism is provided that resolves the reference by searching for an `AbstractContent` whose name attribute of type `String` matches the given input.

Import clauses are not restricted to import single Modelica classes but also a set of sub-classes inside a package. This is indicated by the use of a wildcard `"*"`. Using a reference to `AbstractContent` does not respect the fact that only packages can be referred to when using wild cards. In this definition we do not distinguish between Modelica classes, models, packages and so forth on the syntax level. Therefore the correctness of the rule has to be regarded by the semantics of Modelica. This is surveyed in section 3. Finally an optional comment adds additional information about the import for developers.

Altogether the grammar definition consists of over 100 rules. Performance issues during parsing prevented us from reusing the language definition from the Modelica language specification [9], so that we were forced to create an optimized version. Moreover, the complexity of the resulting AST would have made it difficult to modify or investigate the parse results. As an example, the definition of expressions has been simplified. We do not distinguish between logical expressions, terms, or factors but only define a single type of expression. If needed, the type can be derived by investigating the operator of an expression. This enabled us to reduce the number of rules to 4 compared to 12 defined in the Modelica language specification [9].

2.2 Project structure definition with EMF

For the sake of reuse, Modelica documents should be structured in projects. When defining e.g. a wind energy plant, every component like `Tower`, `Nacelle` or `RotorBlade` ought to be encapsulated in its own unit. If the components are divided into different projects,

they can be interchanged easily. This helps the developer to survey the structure of the designed physical systems. Furthermore, the reuse of functionality from libraries like the Modelica Standard Library is essential.

Separating components into projects requires a mechanism that enables linking between models inside separate projects. A `WindTurbine` e.g. reuses the component `Tower` for the definition of a new wind turbine. Therefore a link between the wind turbine's definition and the document where the tower is defined in has to be established. This ensures the existence of the reused component and enables the user to quickly display the tower's definition. In order to be able to find this kind of references quickly, meta data must be provided that holds additional information about the class structure and location of Modelica files. This data structure is defined with EMF. The central data in this structure is called `ModelResource`. A `ModelResource` can contain source folders. All Modelica source files contained in these source folders belong to the same `ModelResource`. When a source file is parsed, the internal structure is analyzed and stored in an index file. These files are again coupled to the `ModelResource`. Hence `ModelResources` know the name space of all contained models and allow quick linking by the use of qualified names. Qualified names distinctly address a component inside a name space like in `Modelica.SIunits.Angle` whereas `Modelica` and `SIunits` represent packages and `Angle` is a type definition inside this package.

At first sight the proposed project structure looks quite similar to the one Eclipse uses for plug-in projects. In fact many concepts are reused but also altered to meet our requirements (see figure 2). Using `ModelResources` instead of projects as management unit allows to have several name spaces inside one project. This is important when simulations are performed. The source files of several simulations can be kept in the same project and allow to keep track of source code changes between different simulations. In the project based approach it would be impossible to keep multiple class definitions with the same qualified name. To enable linking, every experiment gets its own `ModelResource` that knows the files used for simulation. The `ModelResource` mechanism is also used to enable referencing other projects in the workspace. Projects can be exported as compressed and possibly encrypted libraries. The meta data are kept inside the archive, therefore no further analysis of the contained source files is needed when libraries

are reused. The creation of libraries serves two purposes: First it allows to assemble specific functionality into one library that can be shared among users or maybe even sold to customers; Secondly, using libraries speeds up the development process because handling of a huge number of source files can lead to a slow development environment and an increasing reaction time of the systems on user interaction.

Using EMF turns out to be very helpful, because the data defined in EMF automatically has a persistence model. Also references between EMF-based files are resolved automatically. Furthermore it provides a system for change notification that allows to react on any changes of the meta data.

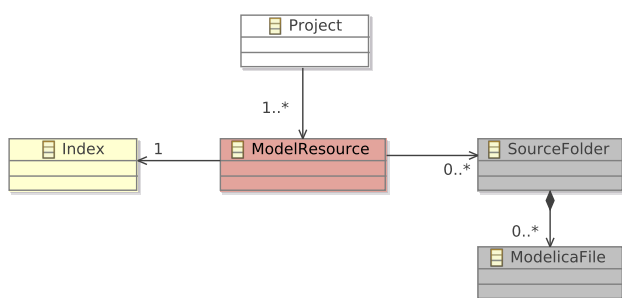


Figure 2: Data structure of the Modelica IDE

3 Verification of Modelica documents

Enhanced verification of code is desired during development to immediately ensure the correctness of the created models. Two kinds of verification can be distinguished - dynamic and static verification.

3.1 Dynamic versus static verification

Dynamic verification of Modelica models is, at the moment, a difficult topic because for this issue models have to be interpreted on instances. For instance, to ensure that a parameter does not exceed a specified value, the equations altering the variable must be calculated. But currently these calculations are done by translation of Modelica source code to a different programming language like C++ and execution of the resulting program code. That means no interpreter working directly on Modelica source code is available but the code is transformed to another kind of programming language and then executed.

The same problem is faced when trying to debug Modelica code like it is done in other object oriented languages, e.g. Java. It may be possible to implement

an interpreter for basic Modelica language constructs and simple models that do not contain any equations. But at the moment there is no solution available that solves Differential Algebraic Equations (DAE)'s during development time.

The MDS community is, by the way, facing the same problem. Instead of generating code that has to be executed, interpreters are often desired that create functionality based on objects. Therefore solving the problem of interpreting models directly could solve the problem of debugging and verification as well as reduce the required time during development.

However static verification is currently done for a wide range of models (e.g. Modelica, UML, Java, ...). Several techniques are available for the verification of models. One could write Java-Code or utilize specialized languages like Check, that is delivered with the Xtext-Framework. In our IDE we use OCL [2] which is a standard language of the OMG. It is spread in research and industry and thus is typically to be understood by many developers. The static verification of Modelica with OCL is presented in the next section.

3.2 Static verification with OCL

In the Modelica specification the semantics of the language is verbally specified. We interpreted the Modelica semantics as Well-Formedness Rules (WFR)s and found 201 WFRs which we translated into OCL constraints. The WFRs differ from each other in terms of complexity. Some constraints are quite easy to define and the execution time is short. Others are complex and often recursive. The complexity and recursivity is conditional upon the underlying Modelica metamodel and OCL as a language that specifies navigation paths through a model. This can cause the verification to take a long time because large parts of the AST have to be considered during the interpretation of the constraints. In the following OCL will be shortly introduced. Then the definition of some WFRs is explained.

3.2.1 OCL

OCL is a language that allows the definition of constraints on objects. Originally it was designed to enable more precise UML diagram definitions. Later OCL has been extended to a query language and is generally in metamodeling. Figure 3 displays a simple example of an UML class where the attribute age inside the class Person is constrained. The invariant

ensures that the age is higher than 17 to fulfill the requirement of being an adult.

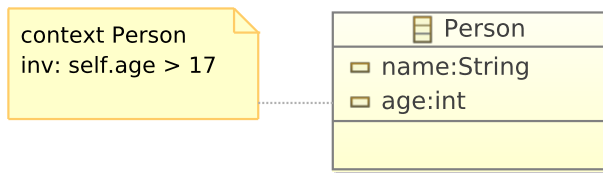


Figure 3: A simple OCL example

Besides invariants OCL also allows to define queries to collect objects. Furthermore the definition of new or derived model elements (attributes, associations, operations) is allowed and helpful to reuse common functionality. OCL also defines a standard library that provides basic functionality like operations on collections (e.g. checking if a set is empty). The language description and examples can be found in the OCL specification [2].

3.2.2 Modelica and OCL

As mentioned above, OCL is not restricted to be used for UML diagrams but can principally be applied to arbitrary object structures [5]. The only restriction is, that the interpreters need to be able to handle the constrained model. This usually means that the meta model of the restricted language must be available and the interpreter must be able to read the data format. As we use Xtext for the definition of our Modelica language, the resulting AST is based on Ecore. Hence an Ecore based OCL interpreter is needed. Therefore two popular OCL interpreters that fulfill this requirement were evaluated for verification, Eclipse OCL⁶ and Dresden OCL⁷ [13].

The basis for the verification of Modelica code is the AST of a Modelica document. It is created by the parser that Xtext generates based on the grammar definition. Thus the AST represents the structure of the Modelica document and can be used to check whether the structure is correct. OCL constraints are defined and evaluated on the AST.

The OCL constraints were used to measure the performance of both tools mentioned above. Furthermore the constraints were analyzed to find time consuming rules. This is important when dealing with user interfaces because users do not accept lags when editing documents because of verification tasks that are per-

	non-recursive constraints	recursive constraints
constant time	x	-
linear time	x	x
quadratic time	x	x
exponential time	-	x

Table 1: Classification of OCL constraints by their complexity

formed in the background. Two categories with different complexity and calculation times were detected (Table 1).

Short examples for non-recursive constraints will illustrate the definition of Modelica WFR in OCL. Listing 2 represents a constraint that can be evaluated in constant time:

```

inv predefined_string_type :
name <> 'String'
    
```

Listing 2: Non-recursive OCL with constant time constraint

Each component declaration in Modelica has a name. In our grammar this is reflected as a rule `ComponentName` with the attribute `name`. Because Modelica reserves some names for components (i.e. `String`), these names are not allowed for newly defined components. The OCL invariant given in Listing 2 ensures that the name `String` is not assigned to a component. As no other objects have to be considered, the calculation depends only on the object `ComponentName` resulting in a constant calculation time.

A linear non-recursive constraint is defined in Listing 3:

```

context Component
inv component_name_type :
not componentnames->exists(
name = self.type.name)
    
```

Listing 3: Non-recursive OCL constraint with linear time

Figure 4 shows the result of our validation in the editor and the problems view.

Components in Modelica classes must not have the same name as their type. It is e.g. prohibited to define a component `Angle Angle;`. The constraint in Listing 3 checks whether a name exists (`componentnames->exists()`) that is equal the name of the components type (`self.type.name`). The calculation time coheres directly with the num-

⁶<http://www.eclipse.org/modeling/mdt/?project=ocl>

⁷<http://www.reuseware.org/index.php/DresdenOCL>

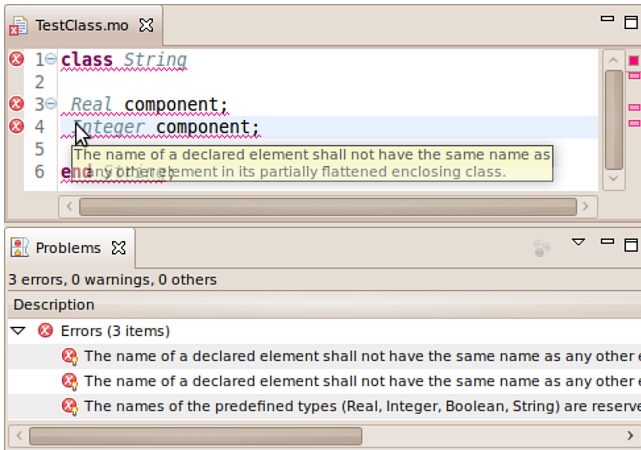


Figure 4: Displaying verification errors

ber of instances that are defined in the component and thus increases linearly.

Another problem is the test for uniqueness which results in quadratic calculation time. If uniqueness of element names in an enumeration shall be ascertained, all elements have to be compared with each other (as long as the model elements are not stored in a relational database). Thus the calculation time grows quadratically to the number of elements. The corresponding OCL constraint is defined in Listing 4:

```

context EnumerationList
inv unique_enum_literals :
enumerationliterals ->isUnique (
componentname.name)
    
```

Listing 4: Non-recursive OCL constraint with quadratic time

Most of the OCL constraints defined for the verification of Modelica documents are of recursive nature because most WFRs originate in restrictions on inheritance structures. As a result many parts of the model have to be considered for verification. Although the effort of calculating non-recursive rules may result in quadratic time as seen above, recursive rules are even worse.

For complexity reasons, only one recursive constraint with exponential time is explained in this paper (Listing 5). The Modelica specification defines the rule: “The type prefixes flow, input, and output shall only be applied for a structured component, if no elements of the component have a corresponding type prefix of the same category.” [9] The function definition in Listing 5 returns a Boolean value that indicates, whether a Modelica class contains a component definition with the prefix flow, input, or output (the collection of components is defined in another func-

tion collectIOComponents()). Not only the analyzed class has to be considered for this constraint, but all super classes from which the instance inherits. This requires the invoking of the same function containsIOPrefixes() recursively and results in the complexity $\mathcal{O}(n^r)$ where n describes the number of super classes and r the recursion depth.

```

context AbstractModelicaClass
def: containsIOPrefixes() Boolean =
collectIOComponents()->size >0
or
collectExtendsClauses()->exists (
containsIOPrefixes)
    
```

Listing 5: Recursive OCL constraint with exponential time

Because of the complexity of the resulting constraints not all WFRs from the Modelica specification have been implemented yet. In a first version of the Modelica IDE (Section 4 we decided to integrate Eclipse OCL because of its better interpreter performance. In addition to the rules from the specification, further constraints may be helpful for daily work. E.g., warnings could be displayed if too many subclasses in a document exist or the package structure is too deep. This functionality may be integrated in a later version of our IDE.

4 Modelica IDE

In this section the main parts of our IDE, which supports the developer in the creation and manipulation of models are presented. First the features of the generated and enhanced Modelica editor are presented (Section 4.1), then it is explained how additional views simplify the development process (Section 4.2).

4.1 Editor

Modern Modelica IDEs support the user in the development process by providing editors and related tools that ease the handling of big projects. Editors provide syntax highlighting to emphasize language specific keywords and to reveal the structure of the written code, making it easier for the developer to understand the code. Highlighting and the recognition of syntactical errors is provided by Modelica specific lexers and parsers that can either be hand written or generated by tools like ANTLR as described in Section 2.1. Furthermore semantical highlighting may be provided but should only be checked where the calculation can be done quickly. In Figure 5 the simple data type Real

(gray, italic) is highlighted semantically while all other decorations are provided by the generated lexer. Code folding allows to reduce the complexity of the displayed code, e.g. by hiding annotations that contain arbitrary information. With the integrated mouse-over help, details about utilized classes that are defined as comments in the declaration can be explored by the developer.

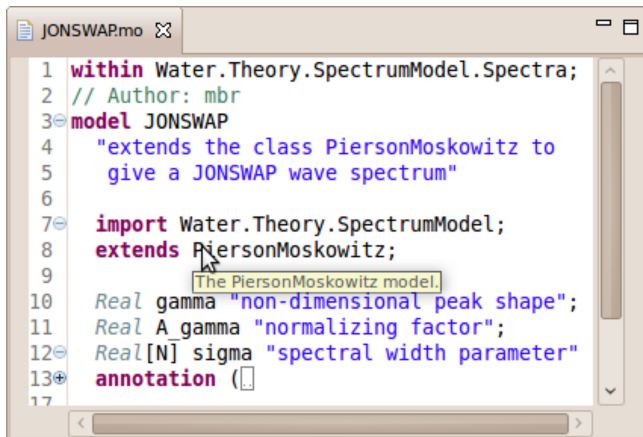


Figure 5: Xtext Modelica editor

Based on the index data, code completion supports the developer in choosing sub-components of classes. This is helpful during the definition of import clauses or components and helps to speed up the development process significantly.

Error markers for non-existent referenced classes are automatically created. This is an advantage compared to editors like the ones integrated in Dymola, or Modelica Development Tooling⁸ because developers immediately recognize these kind of failures. A quick fix using a comparable mechanism as code completion tries to provide a suitable solution.

4.2 Views

The Modelica IDE has several views that display additional information on the projects data. The main view is the Eclipse project explorer that has been extended to fulfill the needs of a Modelica IDE. The contained packages and classes of Modelica files as well as the package structure of referenced libraries are displayed as shown in Figure 6. Each of these classes can be inspected in the editor whereas library documents are opened read-only to avoid the modification of the source code. References inside the documents are linked to allow quick browsing through the source code and the exploration of class declarations. The

Eclipse outline view displays the inner structure of the opened document. All actions that are triggered in the user interface are implemented as Open Services Gateway initiative (OSGi)⁹ events. Together with an Xtext based DSL that was specified for scripting purposes, we are able to record these actions and save them in a file. The engineer can edit or create a script document for automatic execution of actions. This includes the simulation of models with the coupled solvers Dymola and Mosilab [1].

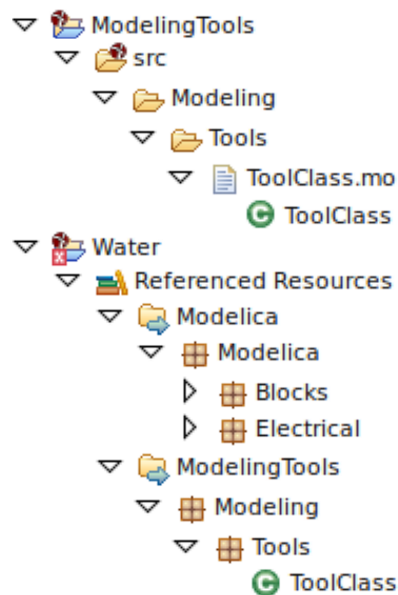


Figure 6: Modelica Project Explorer

5 Conclusion and future work

With the help of MDSD techniques we were able to implement a Modelica IDE in a short period of time. Generated code ensures high quality products and reduces the implementation time by automatic creation of frequently used components like data structures and serialization mechanisms. Since the generated code is based on the same data description (Ecore), interoperability is guaranteed. Therefore augmenting the generated editor with additional functionality became easy.

The introduced IDE enables the Modelica developer to create models fast and easy. The verification and referencing mechanisms ensure correctness throughout the development process. The views on the Modelica data structure help surveying large projects. Encapsulating source code into libraries speeds up the user interaction and encourages the developer to create

⁸<http://www.ida.liu.se/~pelab/modelica/OpenModelica/MDT/>

⁹Open Services Gateway initiative, <http://www.osgi.org>

models as components. This helps in creating frameworks that can be shipped and reused as libraries.

During the development some performance issues arose that should be considered in the future. This became evident when the Modelica Standard Library was imported as a source project. The low performance of parsing all library files is engendered by the complex Modelica grammar that causes a lot of back-tracking during parsing. Allowing the use of a different parser generator than ANTLR to generate a LR(k) parser might help solving this issue. Also future modifications of the grammar definition might speed up parsing, e.g. by defining a unique start and end terminal sign for annotations. In many documents the percentage of annotations compared to executable code is very high. These parts of code should only be parsed if they are needed for example when visualizing the models. Thus a second optimized parser could be introduced for annotations.

Performance issues are a big problem in our approach. Comparing the parser generated by Xtext with the one of the Modelica SDK[11] [12] which is based on the same parser generator technology (ANTLR) might be useful in finding the reasons for these problems. Furthermore the mechanisms of verification should be compared regarding completeness and performance.

A nice feature to have is an editor that allows the developer to compose models from components graphically, like Dymola does. As we use Ecore as basis for our data, the Graphical Modeling Project Graphical Modeling Project (GMP)¹⁰ might be a suitable solution for this task. However, the embedding of graphical information inside annotations of the Modelica documents instead of separate files might cause problems when using GMP. In our opinion, layout information and executable code should be separated as both are independent concerns.

Furthermore, the refactoring of Modelica models should be addressed in the future. Many of the refactorings introduced in [7] would be helpful in Modelica, as it is suitable for most object oriented languages. In [6] an impressive way of source code refactoring by role definitions is explained and demonstrated based on EMFText¹¹. At the moment serializing with Xtext is error-prone and therefore prevented us from integrating the refactoring tool into our project. This will be done as soon as the serialization problems are solved.

¹⁰<http://www.eclipse.org/modeling/emf/>

¹¹<http://www.emftext.org/index.php/EMFText>

References

- [1] J. Bastian, O. Enge-Rosenblatt, P. Schneider: MOSILAB - a Modelica solver for multiphysics problems with structural variability. Conference on Multiphysics Simulation - Advanced Methods for Industrial Engineering, January, 2010, Bonn, Germany
- [2] The Object Management Group (OMG): OCL 2.2 Specification. 2010, <http://www.omg.org/spec/OCL/2.2>
- [3] T.J. Parr, R.W. Quong: ANTLR: A Predicated-LL(k) Parser Generator. *Software | Practice and Experience* 25(7) (1995) 789-810
- [4] F. Budinsky, S.A. Brodsky, E. Merks: Eclipse Modeling Framework. Pearson Education, 2003
- [5] M. Seifert, R. Samlaus: Static Source Code Analysis using OCL. In: Proceedings of the Workshop OCL Tools: From Implementation to Evaluation and Comparison, OCL 2008, Satellite event of the 11th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2008), September 28 - October 3, 2008, Toulouse, France
- [6] J. Reimann, M. Seifert, U. Assmann: Role-Based Generic Model Refactoring. In: *Lecture Notes in Computer Science (LNCS 6395) - Model Driven Engineering Languages and Systems*, Springer, 2010, 78-92
- [7] M. Fowler: *Refactoring: Improving the Design of Existing Code*, Addison-Wesley, Boston, MA, 1999
- [8] D. Steinberg, F. Budinsky, M. Paternostro, E. Merks: *EMF: Eclipse Modeling Framework*, Addison-Wesley, 2009
- [9] Language Specification, Modelica - A Unified Object-Oriented Language for Physical Systems Modeling Version 3.1, May, 2009, <https://www.modelica.org>
- [10] T. Parr: *The Definitive ANTLR Reference: Building Domain-Specific Languages*, Pragmatic Bookshelf, May, 2007
- [11] M. Tiller: Parsing and Semantic Analysis of Modelica Code for Non-Simulation Applications, In: Proceedings of the 3rd International

Modelica Conference, November 3-4 2003,
Linköping, Sweden

- [12] P. Harman, M. Tiller: Building Modelica Tools using the Modelica SDK, In: Proceedings 7th Modelica Conference, September 20-22 2009, Como, Italy
- [13] M. Krebs: Verifikation von Modelica-Programmen mit OCL, Diploma thesis, TU Dresden 2010

Tool Support for Modelica Real-time Models

Michaela Huhn¹, Martin Sjölund², Wuzhu Chen¹, Christan Schulze¹, and Peter Fritzson²

¹Clausthal University of Technology, Department of Informatics
Julius-Albert-Str. 4, 38678 Clausthal-Zellerfeld, Germany

²Linköpings Universitet, Dept. of Computer and Information Science
SE-581 83 Linköping, Sweden

Abstract

The challenges in the area real-time simulation of physical systems have grown rapidly. To prepare a simulation model for execution on a real-time target, an experienced developer usually performs several adaptations on the model and the solver in order to reduce runtime and communication needs.

Two-folded tool support for evaluating the effect of such adaptations is presented here: (1) A *ModelComparator* for the systematic comparison of simulation results from different versions of the model and (2) an *RT-Profiler* for measurements and analyses of function calls during RT simulations. The *ModelComparator* facilitates verification of a model adapted for real-time execution to ensure that it will produce sufficiently accurate results at selected operation points. The *RT-Profiler* takes the specific code structure of simulation models into account when measuring execution times. It directs the developer to those parts that are most promising for model adaptations.

We consider OpenModelica and SimulationX as modeling and code generation frameworks for real-time simulation. The procedure of model adaptations and the use of the analysis tools therein are exemplified in small case studies.

Keywords: simulation, Modelica, RT-profiling, optimization, hardware-in-the-loop

1 Introduction

Modeling and simulation has become an essential part of the design process in mechanical engineering and mechatronics. The object-oriented language Modelica is widely accepted for physical modeling in many industries. Simulation has been applied for analysis and validation in the concept and rapid prototyping phase for a long time, but nowadays simulation models are re-used in latter design phases for testing and verifi-

cation or even as part of the running system. The potential of simulation models in later design phases is to partially substitute costly physical components or complicated conditions of the surroundings that are required to verify the control of a complex system within its mechatronic environment. However, these usages are most often based on online simulation. Hence, simulation models need not only reflect the physics of the modeled components properly from a functional viewpoint, but also with respect to their timing behavior. Thus, the interest in real-time simulation is increasing rapidly.

Prominent usage scenarios for real-time simulation of physical systems are Rapid Control Prototyping (RCP) and Hardware-in-the-Loop (HIL)[12] where the real-time simulation substitutes mechatronic components during detailed design and verification of embedded controllers. Another usage is Model Predictive Control (MPC)[12] where the model becomes a part of the controller used for predicting the short term behavior of a physical component.

The challenge of simulating a Modelica model as part of a real-time system is to meet the timing constraints and meanwhile keep the result accuracies and resource consumption in an acceptable range. Especially when simulating in a hard real-time (HRT) context, any violation of timing constraints during the simulation may cause a fatal failure of the whole system. Since the HRT case is more critical than a soft real-time (SRT) case, it is worth the subject in this paper.

We present two tools to assist the modeler with the validation and verification of real-time simulation models which may be used independently from any specific Modelica framework: The *ModelComparator* aims for verifying whether a model optimized for real-time execution will operate with an acceptable accuracy by comparing its outputs to a reference model. On the other hand, the *RT-Profiler* will aid the modeler

to understand better the timing behaviors and the internal complexity of the model. In profiling the code for measurement of execution times, call frequencies or resource consumptions will be inserted into the model at places most interesting, which is known as instrumentation. As a consequence, instrumentation will add extra instructions to the model and hence increase the total execution time of it. So in general, the overhead caused by instrumentation should be kept in a minimal extent.

The rest of the paper is structured as follows: In Sec. 2 some techniques for adapting models for real-time simulation are sketched. In the Sec. 3 the *ModelComparator* is presented, whereas *RT-Profilier* is described in Sec. 4. Sec. 5 illustrates tool usage in small case studies. Section 6 concludes.

2 Strategies to adapt a model for real-time execution

A key characteristic of HRT systems is time-determinism. Thus, the first requirement on simulation models for real-time execution is to ensure predictable timing behaviors. The second requirement is that the simulation model shall react as fast as the original (physical) system would do, which means that the execution time must not exceed the step size.

Several Modelica simulation environments are capable of exporting Modelica models for real-time simulation already, to name but a few, SimulationX[®], Dymola[®], OpenModelica, etc. In many cases, models for real-time simulation are derived from off-line simulation models developed in an earlier design phase. Such adaptations on a model are called real-time optimization (RTO) of the design model. Some parts can be automated, but due to the diversity of model domains and simulation goals, model engineers commonly need to manually optimize the models to enhance time determinism and performance for RT simulations [2].

The numerous variants to be performed on off-line models can be roughly categorized into two groups:

- RTO by adapting the system behaviors
- RTO by mathematical reduction of complexity

2.1 RTO by adapting the system behavior

Hybrid dynamic systems can be adequately described as a set of nonlinear differential algebraic equations

(DAEs) having a general implicit form as:

$$0 = f(x, \dot{x}, y, z, u, p, t) \quad (2.1)$$

$$0 = g(x, y, z, u, p, t) \quad (2.2)$$

with

x	continuous state variables
\dot{x}	time derivative of x
y	outputs of the system
z	discrete state variables
u	inputs of the system
p	parameters
t	time

where equation (2.1) gives the evolutionary rule of the state variables inside the dynamic system and equation (2.2) implies the algebraic constraint of the system. According to *Implicit Function Theorem*, assuming $D_y g$ (Jacobian of g) is not singular, $\exists \phi(x, z, u, p, t) = y$. Inserting this result into equation (2.2) then gives the following DAEs:

$$0 = f(x, \dot{x}, \phi(x, z, u, p, t), z, u, p, t) \quad (2.3)$$

$$0 = g(x, \phi(x, z, u, p, t), z, u, p, t) \quad (2.4)$$

However, the above mentioned equation system is still in an implicit form. Symbolic analysis and possibly associated manipulations for the index reduction need to be performed on the DAE system to produce an explicit ODE system. Although this can be automatically processed either by commercial Modelica compilers or free ones, some non-linear implicit relations often remain on the RHS of the ODE system. These mathematically complex interdependencies between state variables cause algebraic loops that significantly contribute to execution times during simulation. Breaking up algebraic loops, i.e. decoupling state variables, decomposes the system model into a set of subsystems that is computationally simpler to handle. We briefly sketch some strategies for decoupling:

The most straightforward approach is to eliminate state variables which are of minor relevance to system dynamics, like for instance the fluid temperature in a 1-D momentum equation. Library elements offering different model variants of the same physical entity support the modeler with this strategy: The variants, which may be selected via parameters, may be optimized for the calculation of different sets of input/output variables. Such variants may exploit specific solvers for subproblems or differ wrt. numerical

accuracy or completeness of the physical effects being modeled. All this will result in differences in the computational complexity and the real-time behavior.

As described in [3], knowledge on weak dynamic interactions between system parts may be used to decouple the system by introducing pre-announced weak dynamic state variables which are solved by implicit integration. Such a decoupling leads to strong improvements on the Block Lower Triangle (BLT) transformation, since the off-diagonal entries will decrease inside the BLT structure.

An another alternative is the mixed-mode integration approach described in [11], which tries to partition an entire dynamic system into separate fast/slow subsystems by analyzing the eigenvalues of the system. In contrast to weak dynamic decoupling, mixed-mode integration can be fully automated.

It has to be noted that all these approaches require a modeler with strong backgrounds on the problem domain, as they are only applicable under certain constraints, e.g. a loose coupling system, a reasonable difference between subsystem dynamics, etc.

2.2 RTO by mathematical reduction of complexity

A system's complexity is characterized by the number of its state variables (or number of dimensions of the model). Some state variables de facto dominate the dynamic behavior of the system, which gives us a possibility to reduce the complexity of the system by disregarding some subordinate ones. Provided that a DAE system has been transformed into the following continuous generalized state-space form:

$$\mathbf{E}\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \quad (\text{state equation}) \quad (2.5a)$$

$$\mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u} \quad (\text{output equation}) \quad (2.5b)$$

with $\mathbf{x} \in \mathbb{R}^n$ (state variables), $\mathbf{u} \in \mathbb{R}^m$ (inputs), $\mathbf{y} \in \mathbb{R}^l$ (outputs), $\mathbf{E} \in \mathbb{R}^n \times \mathbb{R}^n$ (descriptor matrix), $\mathbf{A} \in \mathbb{R}^n \times \mathbb{R}^n$ (system or state matrix), $\mathbf{B} \in \mathbb{R}^n \times \mathbb{R}^m$ (input matrix), $\mathbf{C} \in \mathbb{R}^l \times \mathbb{R}^n$ (output matrix) and $\mathbf{D} \in \mathbb{R}^l \times \mathbb{R}^m$ (feed-through matrix).

In (2.5), if the number of state variables is very large compared to the number of inputs and outputs ($n \gg l, n \gg m$), it implies that redundancies might exist in the system. Hence, there is a chance to come up with an alternative system model not only with much lower dimensions but also with adequate accuracies and the preservation of important system properties. This technique is called *model order reduction* (MOR).

A brief introduction of the so-called projection-based model order reduction (PBMOR) will be given here to show the philosophy of MOR. The PBMOR can be roughly performed in three steps:

1) Choice of ansatz:

The state variables $\mathbf{x} \in \mathbb{R}^n$ is approximated by $\mathbf{x}_q \in \mathbb{R}^q$ ($q \ll n$):

$$\mathbf{x} \approx \mathbf{x}_q = \sum_{j=1}^q \mathbf{v}_j \xi_j = \mathbf{V}_q \boldsymbol{\xi} \quad (2.6)$$

where \mathbf{v}_j is the basis spanning a subspace $\mathcal{V}_q = \text{span}\{\mathbf{v}_1, \dots, \mathbf{v}_q\}$ called ansatz space of the reduced model, $\boldsymbol{\xi} \in \mathbb{R}^q$ and the matrix $\mathbf{V}_q = [\mathbf{v}_1, \dots, \mathbf{v}_q]$ ($q \ll n$). The choice of decent basis \mathbf{v}_j plays a significant role in PBMOR, there exist many methods on the market [8, 5].

2) Insert the ansatz:

Inserting the ansatz (2.6) into (2.5) delivers the following over-determined system:

$$\mathbf{E}\mathbf{V}_q \dot{\boldsymbol{\xi}} = \mathbf{A}\mathbf{V}_q \boldsymbol{\xi} + \mathbf{B}\mathbf{u} \quad (2.7)$$

$$\mathbf{y} = \mathbf{C}\mathbf{V}_q \boldsymbol{\xi} + \mathbf{D}\mathbf{u} \quad (2.8)$$

As there are more equations than unknowns in this reduced model representation, in general the residual of the system will not equal zero.

3) Projection of the residual:

The residual is projected onto a subspace \mathcal{W}_q spanned by the columns of so-called weighting factors \mathbf{W}_q .

$$\mathbf{W}_q^T \mathbf{E}\mathbf{V}_q \dot{\boldsymbol{\xi}} = \mathbf{W}_q^T \mathbf{A}\mathbf{V}_q \boldsymbol{\xi} + \mathbf{W}_q^T \mathbf{B}\mathbf{u} \quad (2.9a)$$

$$\mathbf{y} = \mathbf{C}\mathbf{V}_q \boldsymbol{\xi} + \mathbf{D}\mathbf{u} \quad (2.9b)$$

Thus, an optimal solution of the over-determined system can be attained in a least square sense. The input-output properties of the reduced model described in (2.9) depends only on matrices \mathbf{V}_q and \mathbf{W}_q .

2.3 Solver

In a real-time setting the well-established Euler Forward solver is considered most suitable for time integration. As an explicit fixed step solver it guarantees a limited number of calculations and thereby predictable execution times. Moreover, it offers acceptable inaccuracies near discontinuities [2]. If the system contains any non-linear behavior, then iterative solvers have to be employed for solving nonlinear equations. As a matter of fact, this will lead to unpredictable execution time, which is not preferable in real-time simulation. The execution time when applying an iterative solver depends on the number of iterations needed for

the solution to converge. Setting up an upper bound for the iteration number will be a work-around to ensure the predictable timing behavior, but such treatment might induce numerical inaccuracies and needs to be taken in to account carefully. Thus, the two central issues in real-time simulation, timing behavior and accuracy, are reflected by the *ModelComparator* and *RT-Profiler* in this paper.

3 Comparing model variants

In many cases, several RTO techniques have to be applied on a compound model to finally meet the real-time constraints. However, simulating in time is just a necessary but not a sufficient condition for RT simulations, because simulation results with instabilities or large deviation errors will not make any sense. So after having performed several RTOs on a model, the quality of simulation results has to be assured. This can be done with the help of the *ModelComparator* by comparing the simulation results between the original and optimized models. Although many Modelica simulation environments - like SimulationX, Dymola and OpenModelica - already offer the functionality for comparing results for variants, these are limited in that all variants must be modeled and simulated in the same framework. The *ModelComparator* has however surmounted this limit and it is capable to compare results from different frameworks. If the new Functional Mock-up Interface (FMI) [1] standard will be widely adopted by Modelica tool providers for model export, the *ModelComparator* can be further extended for direct manipulation of the model on source code level.

Moreover, the *ModelComparator* helps the developer to make a choice between a model leaning more towards the RT performance and a model leaning more towards the accuracy in a concrete application context, for instance, the anti-lock breaking system employed in automotive industry may prefer a more rapid response of the RT model for HIL testing.

The *ModelComparator* is an application that was developed with Java in Eclipse IDE with GUI support, its outputs are shown in Fig. 8 and 14. The user may load different models and select for each of them variables that will be plotted and compared to each other. As it allows users to handily modify the parameters for simulation and solver settings via the GUI, it is very useful when carrying out a parameter optimization, too. Another feature of *ModelComparator* is to automate series of simulations and analyze them wrt. thresholds: The user may specify threshold values be-

fore starting the simulation that indicate in some sense exceptional behavior. The *ModelComparator* filters the results and directs the user conveniently to those instances of a series of simulations where the values are exceeding the threshold. Currently it supports exported model from SimulationX under Windows, but the scope of environment support will be extended.

4 RT-Profiling for simulation models

The purpose of profiling in the context of real-time simulation is two-folded: (1) RT-profiling is a means to verify whether an optimized model satisfies its real-time requirements and (2) RT-profiling shall allow a detailed analysis to determine those parts of the model causing the major portion of the computational load. Thus, RT-profiling shall support the modeler to identify the primary candidates in further optimization and verification steps. Hence a RT-profiler tailored to the specific code structure of simulation models is preferred to general purpose profiling tools like *prof* [6].

The structure of the C code generated from simulation models was analyzed and the concepts for instrumenting the code for RT-profiling were introduced in [12]. The profiling was performed on the proprietary C code from SimulationX for the assessment of RT performance on RT target SCALE-RT[®] 5.1.4 [7]. However, the concepts can be transferred to other frameworks by adopting the instrumentation accordingly.

In general, source code automatically generated from simulation models has a flat structure for which profiling, i.e. an statistical evaluation of call frequencies and execution times of functions, seems to be sufficient. A model commonly consists of an initialization and a simulation phase. Each phase is split into global solver steps. In a global solver step a series of integration steps are performed followed by the calculations of the output variables. To guarantee the timely delivery of results, either a fixed step solver is employed or the number of iterations is limited¹. Within each part, external functions may be called. In case the algebraic loops cannot be solved analytically, a local solver will numerically compute the solution. So, a global solver step can be described by:

- n_I · integration steps
 - e_I · external function calls
 - c_I · additional calculations
 - a_I · (non-)linear blocks

¹with the well-known consequences on accuracy

- * e_{al} : external function calls
- * c_{al} : additional calculations
- 1 · output of variables
 - e_O : external function calls
 - a_O : additional calculations

To analyze the allocation of execution time within a global step the profiling will measure the execution time for each (non-)linear block and each external function call, as well as for each integration step as a whole and the calculation of outputs. The values are stored separately, and in a post processing step average, variance and the maximum, which is most important for verification of hard real-time requirements, are calculated.

4.1 RT-Profiling SimulationX models

The C code generated by SimulationX from a simulation model is instrumented in a post processing step as follows: Whenever entering or leaving a function or block a time stamp is recorded and a counter is incremented. As the generated C code is well structured, automation of code instrumentation is straightforward. After determining the tokens at which instrumentation has to be placed, template-based code transformation can be realized easily. We used ANTLR [9] and cc65 [13] as two alternatives for this tasks.

4.2 Implementation on a RT Target

Whereas in [12] version 4.1.2 of Scale-RT was used, we now have moved to the current version Scale-RT 5.1.4 which offers improved support for simulation models by providing a framework that automatically embeds a model in a kernel module that iterates the global solver steps. It was already observed in [12] that storing and evaluating the profiling data significantly contributes to execution time. Thus it should not be done by the task executing the model in the Scale-RT real-time kernel, but by a task running in the (non-real-time) user space. Consequently, we applied the producer-consumer pattern and implemented the profiling as two task communicating via a FIFO-buffer (see Figure 1).

For each global solver step of the model, the profiling methods record the following information:

- execution time and frequency of an integration step

- execution time and frequency of each external function call within the integration step that does not reside inside a (non-)linear block
- execution time and number of loops of each (non-)linear block within the integration step
- execution time and frequency of each external function call within each (non-)linear block
- execution time of outputting variables at the end of the current step

4.3 RT-profiling OpenModelica models

The OpenModelica implementation of the code instrumentation was done in the compiler itself, with only slight modifications. The instrumentation is performed by compiling a model with a preprocessor macro set, and running the executable with the time measurement flag. The time measurement uses the real-time clock available on the platform used. All measurements are accumulated using integer math and output at each time step.

The code runs on all platforms supported by OpenModelica and is not limited to RT systems. Profiling is of general interest because small changes to a model may have a large impact on the simulation time. By providing a profiler to both developers and users of a simulation tool, performance issues can more easily be discovered. As a result it should be easier to improve the quality of the tool. By observing the output of the profiler, you can see that in the SimpleNonLinear example (Listing 1), $\sin(x)$ will be called 3 times in each time step. While it is possible to determine the value of x during compile-time or initialization, OpenModelica does not yet perform these optimizations.

Listing 1: Simple non-linear equation

```
class SimpleNonLinear
  Real x = cos(x);
end SimpleNonLinear;
```

The profiling also works for any user-defined function that is called. In the ArrayCall example (Listing 2), tenCos is called 10 times because arrays were not handled properly by the compiler in this case. This means cos is called 10^2 times in every timestep. If the function is inlined, cos is only called 10 times instead.

Listing 2: Binding equation is an array

```
class ArrayCall
  function tenCos
    input Real r;
```

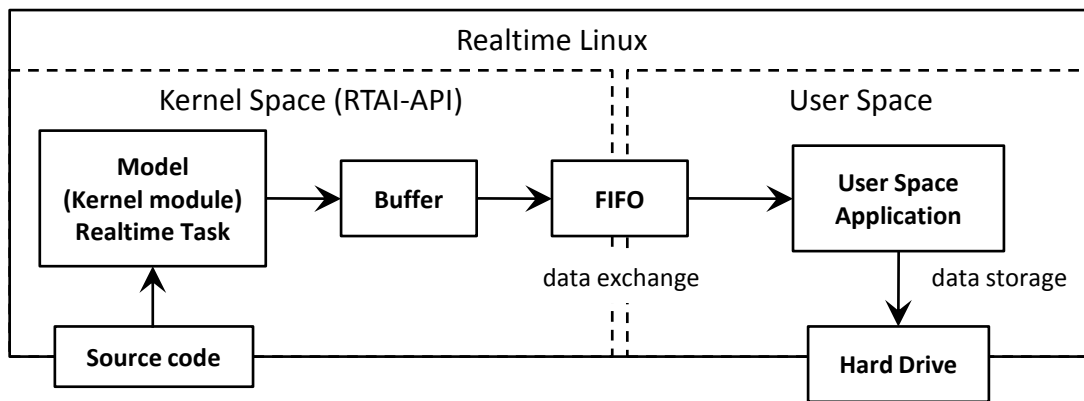


Figure 1: Communication between user task and model task

```

output Real array [10];
algorithm
  array := cos(r*(1.0:10.0));
end tenCos;

```

```

Real x[10] = tenCos(time);
end ArrayCall;

```

The output only contains the number of the equation blocks and names of functions. In the future, this information will be augmented with the names of variables defined by the block and the line numbers where those variables originate from. It is also of interest to add the lines of the equations involved in solving the block since it is hard for a user to understand which of his equations caused a particular error. This is related to the general problem of bug localization in debugging equation-based models [10].

Much of this information is present in the OpenModelica backend, but only part of it is present in the generated code. Once the information is present in the code, the runtime system should be able to generate a detailed report. In the code generated by SimulationX information is present as well, however, it would be the task of the instrumentation to extract and relate it to the profiling results.

4.4 Mapping Profiling Results to Model Positions

Furthermore, additional work is planned for improving the final report that the profiling gives. For example, when displaying the time spent in a non-linear system of equations, the tool should also report the variables involved, what line of code they are defined in. When possible, the tool should also display the line numbers where the original equations were defined.

Much of this information is already available in the

OpenModelica Compiler backend but is not yet propagated all the way into the source code. In the code generated by SimulationX information is present as well, however, it would be the task of the instrumentation to extract and relate it to the profiling results. Adding this information as part of the runtime environment is also of a more general interest to a user since it is hard for a modeler to try and understand which of the equations caused a particular runtime error.

This is related to the general problem of bug localization in debugging equation-based models [10].

5 Case Studies

The case studies presented in this section are going to show the procedures of carrying out RTO on original models, testing model RT performance and validating the optimized models with facilities from RT-Profiler and ModelComparator. It is also shown here how the results from RT-Profiler can guide developers to perform adaptations on design model for a better RT performance. However, the design, modeling and code exporting phases from physical dynamic systems to RT models are not covered in this paper.

5.1 Case 1: Electric circuit with saturating inductors

The first case is a simple electric circuit with inductors showing non-linear behaviors (due to the saturation effect of ferromagnetic materials): It is taken a variant of the basic components from the Modelica Standard Library 2.2.1 [4]. The saturation of an inductor is approximately described by a non-linear function relating the actual inductance with the changes in drive current. The Modelica model is shown graphically in Figure 2, where A and B are the observation points.

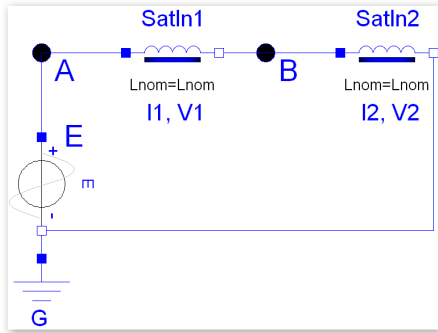


Figure 2: Simple circuit with saturating inductors

5.1.1 System Description

In this model, a time-dependent sinusoidal voltage source E and two nonlinear inductors $SatIn1$ and $SatIn2$ are connected in series. By providing the necessary parameters, e.g. nominal inductance L_{nom} , nominal current I_{nom} , inductance near zero current L_{zer} and inductance at large current L_{inf} , the actual inductance L_{act} can be calculated via $L_{act} = f(I(t))$, where $I(t)$ is the current flowing through the inductor, $f: \mathbb{R} \mapsto \mathbb{R}$ is a nonlinear mapping. From Maxwell Equations it is known that the magnetic flux $\Phi^{(1)}$ and $\Phi^{(2)}$ of the two inductors can be computed through:

$$\Phi^{(1)} = L_{act}^{(1)} \cdot I_1 = f^{(1)}(I_1) \cdot I_1$$

$$\Phi^{(2)} = L_{act}^{(2)} \cdot I_2 = f^{(2)}(I_2) \cdot I_2$$

After manually performing an electric circuit analysis, the following DAEs are obtained:

$$V_A = E \quad \text{Voltage of the source} \quad (5.1a)$$

$$V_A = V_1 + V_2 \quad (5.1b)$$

$$V_1 = \text{der}(\Phi^{(1)}) = \text{der}(f^{(1)}(I_1) \cdot I_1) \quad (5.1c)$$

$$V_2 = \text{der}(\Phi^{(2)}) = \text{der}(f^{(2)}(I_2) \cdot I_2) \quad (5.1d)$$

$$I_1 = I_2 \quad (5.1e)$$

5.1.2 Nonlinearity of the System

The DAE system (5.1) contains highly nonlinear behaviors due to the relation between voltage and current within the saturating inductors. Moreover, the algebraic constraint on V_1 and V_2 forces the system to stay as a holistic system. In order to solve this monolithic (non)linear block, a numerical solver has to be called iteratively, which leads to excessive and indeterministic computation time. A workaround to handle the non-determinism is to limit the maximal number of iterations, but this will sometimes lead to numerical instability. However, the main issue is that it depends

on the tool whether such a workaround is possible or not. So the timing behavior for solving the (non)linear block are of a great interest in RT profiling.

Applying the profiling tool on the model in SCALE-RT 5.1.4 [7] real-time environment yields the results given in the following figure:

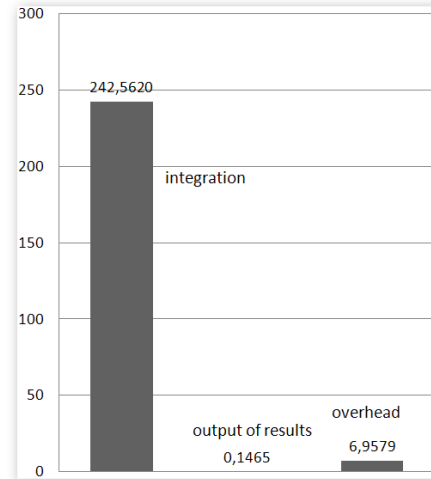


Figure 3: Workload in global steps

It is noticed from Figure 3 that the model runtime of a global solver step is dominated by the number of integration steps and the execution time of each integration step. In this RT simulation, the 10 integration steps contribute more than 97% of the total work load. The time for each integration, $24.2562 \mu s$, is taken as an average of 10 second simulation results. Time for outputs, $0.1465 \mu s$, is relatively small in this case and the overhead caused by auxiliary operations, $6.9579 \mu s$, is small as well. Time for outputs and overhead are also averages of all measured results.

An insight of the workload in each integration step can be achieved by tracing down to the generated source code and through results from the profiling results of each integration step. This is given in Figure 4:

The average execution time for the nonlinear block is $3.7001 \mu s$ and in each integration step the nonlinear solver is called 4.4281 times on the average, so the total time for solving the nonlinear equations is $\tau_{tot_non} = 16.3844 \mu s$. There still exists a linear system after the solution of the nonlinear one is obtained, which is also solved iteratively. The average execution time and the number of loops are $3.5555 \mu s$ and 1.999 times, respectively, in each integration step, which leads to a total time $\tau_{tot_lin} = 7.1106 \mu s$. Compared to the total time for solving nonlinear equations, τ_{tot_lin} is less than half of τ_{tot_non} . Thus, the monolithic

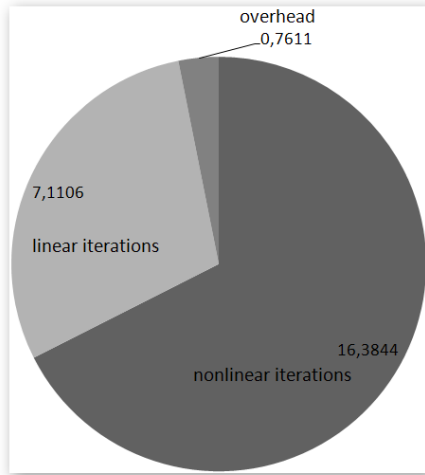


Figure 4: Workload in integration steps

(non)linear block might be a candidate of bottlenecks in the RT model and needs to be dealt with.

5.1.3 RTO by Introducing Capacities

As described above, the nonlinearity of the original system causes computationally expensive algebraic loops during the integration steps. The execution times for solving the (non)linear blocks are more significant than those for other operations. In order to break down the large nonlinear system into smaller subsystems, a capacitor can be introduced into the original system as illustrated in Figure 5. As a result, the original DAE

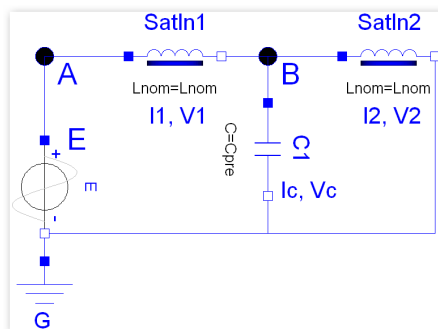


Figure 5: Optimized simple circuit

system (5.1) is transformed as follows:

$$V_A = E \quad (5.2a)$$

$$V_A = V_1 + V_C \quad (5.2b)$$

$$V_1 = \text{der}(\Phi^{(1)}) = \text{der}(f^{(1)}(I_1) \cdot I_1) \quad (5.2c)$$

$$I_C = C \cdot \text{der}(V_C) \quad (5.2d)$$

$$I_1 = I_2 + I_C \quad (5.2e)$$

$$V_2 = V_C \quad (5.2f)$$

$$V_2 = \text{der}(\Phi^{(2)}) = \text{der}(f^{(2)}(I_2) \cdot I_2) \quad (5.2g)$$

Now the voltage V_1 is directly related to the voltage V_C of the capacitor C instead of voltage V_2 . A consequence of this transformation is the decoupling of the dependency between V_1 and V_2 . Thus, the equation system in (5.2) is now decomposed into two smaller subsystems. The profiling results of this decomposed system are shown below.

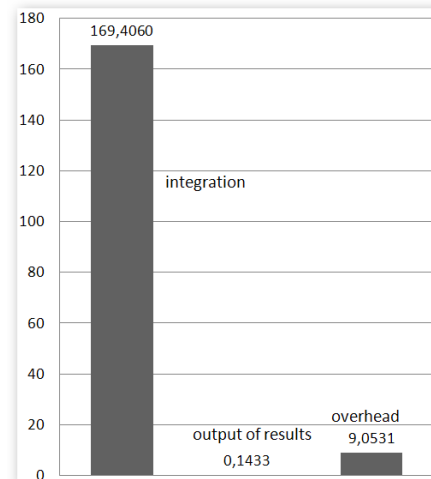


Figure 6: Workload in global steps (optimized)

The results show a decrease of overall runtime in each global step of about 30% workload compared to the original model. The number of integration steps is still 10, but now with $16.9406 \mu s$ average, and together they contribute about 95% of the total workload. The overhead has increased to $9.0531 \mu s$ and the calculation of outputs remains almost the same $0.1433 \mu s$ as given in Figure 6

To check whether the reason for the runtime improvement is the decoupling of algebraic loops, an analysis of the underlying integration steps has been carried out in Figure 7.

From the automatically generated source code it also can be seen there are two nonlinear subsystems. When solving these two nonlinear subsystems, each takes 5.9968 loops and each has execution times of 1.3746

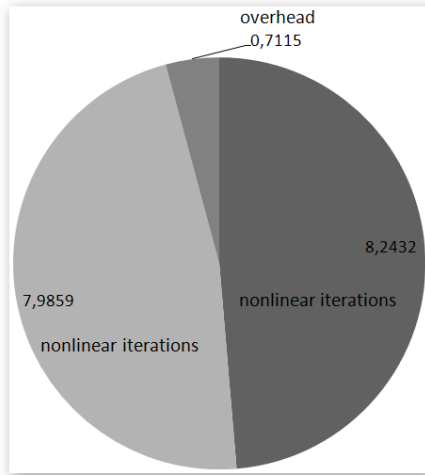


Figure 7: Workload in integration steps (optimized)

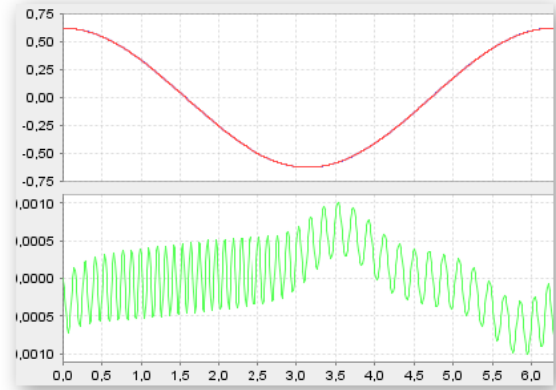
μs and $1.3317 \mu s$. Summing them up, the total time for solving the first nonlinear block is $\tau_{tot_non}^1 = 8.2432 \mu s$, while the total time for solving the second nonlinear block is $\tau_{tot_non}^2 = 7.9859 \mu s$. Comparing these results with the ones shown in Figure 4, although the decoupling of the original system leads to two nonlinear subsystems, which require more iterations, these subsystems are usually easier and more efficiently to be solved. The final effect is an improvement of RT performance.

5.1.4 Deviation Analysis

A side effect of this RTO that introduces a capacitor is an unwanted oscillation which can be observed for instance at V_B . This is because of the nature of capacitor inside a dynamic electric circuit, which is described as a differential equation in (5.2d). Nevertheless, these oscillating errors are so small that it makes almost no difference to the results as it can be seen from the screenshot of the ModelComparator Figure 8.

5.2 Case 2: Nonlinear Thermal Resistor Circuit

An analogue to electric circuit in heat transfer context is a thermal circuit. Consequently the heat flow, temperature, thermal resistance, thermal capacity and temperature source are represented respectively by the current, voltage, resistor, capacitor and voltage source in a thermal circuit. However, any components in a thermal circuit might show nonlinear behaviors. In this case study, the nonlinearities of two thermal resistors are considered.


 Figure 8: Error oscillations of V_B

5.2.1 Case Description

In the thermal circuit given in Figure 9, there are two temperature sources (T1 and T2) providing output temperature consisting of constant offset temperatures and small pure sinusoidal perturbations $T_{out} = T + \sin(t)$. Thermal resistances of the resistors (R1 and R2) are nonlinear functions of the temperature $R = f(T)$, where $f: \mathbb{R}^d \mapsto \mathbb{R}, d \in \mathbb{N}$ is a nonlinear mapping. For instance, the positive temperature coefficient (PTC) and negative temperature coefficient (NTC) thermistor, $R = f(T)$ is an expected physical behavior.

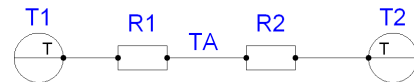


Figure 9: Thermal Circuit with nonlinear resistors

In order to calculate temperature T_A at point A, the following purely algebraic equations have to be solved:

$$q = \frac{T_1 - T_2}{R_1 + R_2} \quad (5.3a)$$

$$q = \frac{T_1 - T_A}{R_1} \quad \text{or} \quad q = \frac{T_A - T_2}{R_2} \quad (5.3b)$$

where q is the heat flow, T_A is the temperature at point A, $R_1 = f(T_1, T_A)$ and $R_2 = f(T_2, T_A)$ are the equivalent thermal resistances of the two resistors. As (5.3) forms a nonlinear equation, algebraic loops are expected in each integration step. The profiling results performed on the code executed on SCALE-RT are given in Figure 10.

The average number of integration steps of a global step is 99.9021 and the average execution time per in-

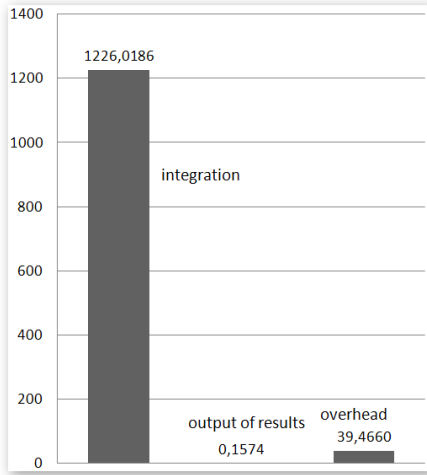


Figure 10: Workload in global steps

tegration is $12.2722 \mu s$. Due to the ascent of the number of integrations per global step, the accumulated overhead has a value of $39.4660 \mu s$. At each global step a calculation of outputs is performed and it takes $0.1574 \mu s$, which is negligible compared to the average runtime $1266 \mu s$ in a global step. From these results, it is obviously to recognize that if a reduction can be achieved on the execution times or number of the integration steps, the RT performance will be enhanced significantly. A detailed view of the workload contribution in every integration step is given in Figure 11. The data in this figure further stresses the fact that the time for solving the nonlinear problem is the bottleneck of this model, which needs to be handled for RT purpose.

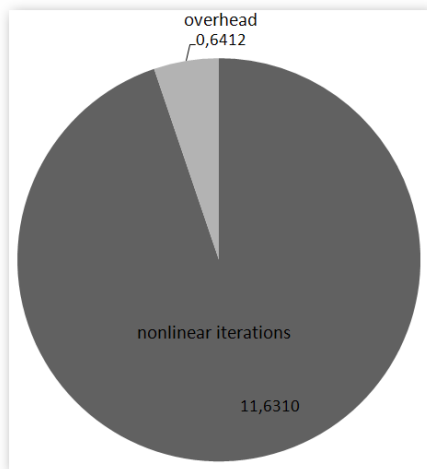


Figure 11: Workload in integration steps

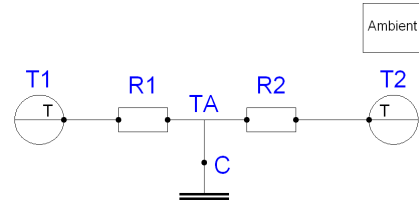


Figure 12: Optimized thermal circuit

5.2.2 RTO by Introducing Capacities

After introducing a thermal capacitor to the original thermal circuit, a decoupling of thermal resistances R_1 and R_2 is obtained (Figure 12). Now the temperature at point A can be substituted by the temperature of the capacitor C instead of solving the underlying nonlinear equation. Assuming $C.T$, $C.C$, $C.\alpha$ and $C.q$ are temperature, thermal capacity, thermal coefficient and heat flow of the capacitor C . q_1 and q_2 are heat flows in R_1 and R_2 . $Amb.T$ is the temperature of the surrounding ambience. Then we have:

$$T_A = C.T \quad (5.4a)$$

$$R_1 = f(T_1, T_A) \quad (5.4b)$$

$$R_2 = f(T_2, T_A) \quad (5.4c)$$

$$q = \frac{T_1 - T_2}{R_1 + R_2} \quad (5.4d)$$

$$q_1 = \frac{T_1 - T_A}{R_1} \quad (5.4e)$$

$$q_2 = \frac{T_A - T_2}{R_2} \quad (5.4f)$$

$$C.q = q_1 - q_2 \quad (5.4g)$$

$$C.C \cdot \text{der}(C.T) = C.q - C.\alpha(Amb.T - C.T) \quad (5.4h)$$

The calculation here is pretty straightforward and no algebraic loops should be observed during a RT simulation, so a performance improvement should be expected. This is proved by the profiling results, since no algebraic loop inside integration steps has been measured. Instead of breaking up the algebraic loops into small ones as shown in section 5.1, the formal existing algebraic loops have been completely eliminated. Although the number of integration steps remains the same as in the original model, the execution time has been drastically reduced to $1.7042 \mu s$. So the average computation time for a global step has been reduced to $219.6088 \mu s$, which can be seen as a triumph of optimizations, see Figure 13.

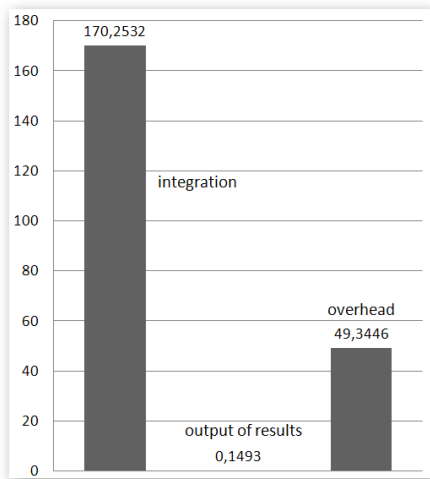


Figure 13: Workload in global steps (optimized)

5.2.3 Deviation Analysis

Model optimizations by introducing an element with some capacity is a good idea to break up the algebraic loops. But as shown in 5.1, one should be cautious when applying this technique on a model. If the time constant of the capacity is less than the integration step size, this will cause instability in a system. Unlike in case 1, where the voltage between those two inductors depends on the time derivatives of the current flowing through, here the thermal resistance R is just a function of the temperature. Hence R can be calculated directly from $C.T$. The following figure gives the comparison of the temperature T_A from original and optimized models.

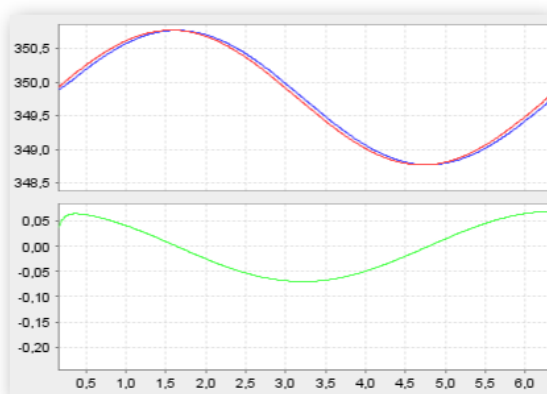


Figure 14: Error oscillations of T_A

As it is shown in the lower diagram, a small time shift of the solution T_A is observed in Figure 14, which causes the error in this case, but still in an acceptable range.

Acknowledgement

The work of the 3d and 4th authors was partially funded by the Federal Ministry of Education and Research (BMBF), Germany, in the projects TEMO (grant 01IS08013C) and OPENPROD (grant 01IS09029D).

We are thankful to Adina Aniculaesei and Mark Wessel for implementation support.

6 Conclusion

Adapting simulation models for execution in a real-time context is often a complex task that requires two-folded verification. First, accuracy of the results obtained with an optimized model and its stability have to be proven. Second, it has to be shown that the real-time constraints are met and if not, which are the most promising parts for further improvements. We presented two tools, a ModelComparator and the RT-Profiling, to support the developer with these tasks and illustrated their usage in two small case studies.

However, an open issue is how to relate the verification results from both, model comparison and RT-Profiling, back to the variables and equations of the optimized model. To solve this issue will be a precondition to enable real-time adaptation of models of another scale of complexity.

References

- [1] MODELISAR (ITEA 2 07006). Functional Mock-up Interface for Model Exchange, January 26 2010.
- [2] T. Blochwitz and T. Beutlich. Real-Time Simulation of Modelica-based Models. In *Proc. 7th Modelica Conference*, pages 386–392. The Modelica Association, 2009.
- [3] F. Casella. Exploiting Weak Dynamic Interactions in Modelica. In *Proc. 4th Modelica Conference*, pages 97–103. The Modelica Association, 2005.
- [4] C. Clauß and A. Schneider. Modelica Standard Library 2.2.1, 2007.
- [5] W.H.A. Schilders et al. *Model Order Reduction*. Springer Verlag, 2008.
- [6] S. Graham, P. Kessler, and M. McKusick. An Execution Profiler for Modular Programs. In

Software - Practice and Experience, volume 13, pages 671–685, 1991.

- [7] Cosateq GmbH & Co. KG. Scale-RT, 2010.
- [8] A.K. Noor. Recent advances and applications of reduction methods. *Appl. Mech. Rev.*, 1994.
- [9] Terence Parr. ANTLR, 2010.
- [10] A. Pop, D. Akhvlediani, and P. Fritzson. Towards Run-time Debugging of Equation-based Object-oriented Languages. In *Proceedings of the 48th Scandinavian Conference on Simulation and Modeling (SIMS' 2007)*, 2007. Göteborg, Sweden. October 30-31.
- [11] A. Schiela and H. Olsson. Mixed-mode Integration for Real-Time Simulation. In *Modelica Workshop 2000 Proceedings*, pages 69–75. The Modelica Association, 2000.
- [12] C. Schulze, M. Huhn, and M. Schüler. Profiling of Modelica Real-time Models. In *3rd International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools (EOOLT)*, pages 23–32. Linköping Electronic Conference Proceedings, 2010.
- [13] Ullrich von Bassewitz. cc65, 2010.

High-speed train pneumatic braking system with wheel-slide protection device : A modelling application from system design to HIL testing

Lionel Belmon, Chen Liu
Global Crown Technology
Lanchoumingzuo Plaza, Chaoyangmenwai Avenue, Beijing, China
lionel.belmon@globalcrown.com.cn, chenl@globalcrown.com.cn

Abstract

Train pneumatic brakes are part of a train safety system, and are thus critical components. This paper illustrates how modeling can be applied to efficiently design such system, from requirement definition to HIL testing. The valves modeling is discussed along with the system level model. Moreover, in order to study the wheel-slide protection device, a model of the wheel-rail interface has been developed.

The contact model, written in Modelica, has been validated against measurement for different conditions of contact (dry, wet...). The model is fully parametric and allows testing of various adherences.

Finally, the resulting system composed of pneumatic valves, wheel-rail interface and rolling-stock is exported through c-code for integration into a HIL system, providing an efficient test platform for the electronic Brake Control Unit.

High speed train; braking ; adherence; pneumatic

1 Introduction

High speed train is under major development in China and a lot of interest is put on the design of subsystems. In particular the pneumatic braking system, which is used for instance in emergency braking, is a critical safety system. Much attention and efforts are dedicated to the robustness and reliability of this system, especially regarding its performance for braking distance.

We introduce the main components of the braking system in Figure 1. The compressor system and the emergency circuit have been omitted of the figure.

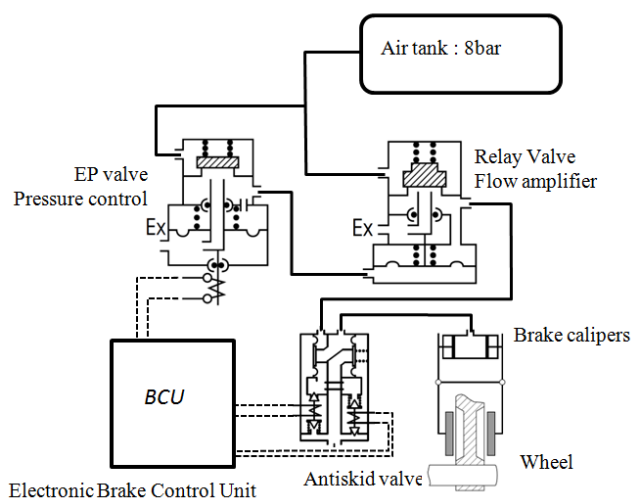


Figure 1 : Simplified schematic of pneumatic brakes [3]

The pressure is supplied by a large air tank. An electro-pneumatic valve (EPV) adjusts the control pressure for the flow amplifier. The flow amplifier valves will work in such a way that the downstream pressure is maintained at the input control pressure. The EPV output pressure is directly controlled by the Brake Control Unit (BCU) and depends on the braking level request.

The Wheel Slide Protection Device (WSP) consists of a set of antiskid valves. These valves are controlled by the BCU in such a way that, when wheel speed is decreasing too fast, the valves modulate brake pressure and prevent wheel blocking.

Besides this basic working principle, an emergency circuit is also available for emergency braking. This circuit has different components and functions but we will focus in this paper on the main braking circuit.

We will now introduce how modeling and simulation supports the design process. We choose the Modelica platform *SimulationX*[®] for its convenient

pneumatic and mechanical libraries. We should also mention the *SimulationX*[®] TypeDesigner tool in which has been quite useful in developing new models in Modelica.

2 Sizing and target pressure

The first step in the design process is to size the system and define subsystems requirements. The EP brake designer will receive as input requirement a target deceleration at a given speed, as shown in Figure 2.

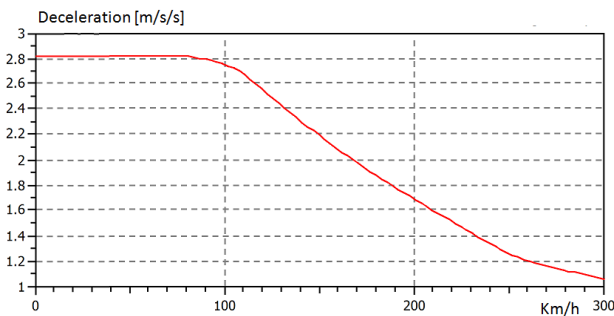


Figure 2 : target train deceleration as a function of speed

The braking force in the train comes partly from the electrodynamic braking but needs to be complemented in certain case by the pneumatic braking system.

Simulation is used at this stage in order to determine the target pressure that the pneumatic brake system should apply to brake cylinders. The model created is a simple model of the rolling-stock, taking into account mass of the cars, rotary inertia of the wheelsets, frictions (aerodynamic...) and electrodynamic brake torque. The brakes model are also simplified but take into account some specific friction effects described later in section 3.4.

The model applies inverse computation in *SimulationX*[®] in order to determine a target pressure to reach the requested deceleration as a function of speed.

The process is illustrated in the Figure 3 and some example of results are provided in Figure 4. The effect of the speed-dependent brake friction coefficient can be clearly seen between in the range [0-100] km/h.

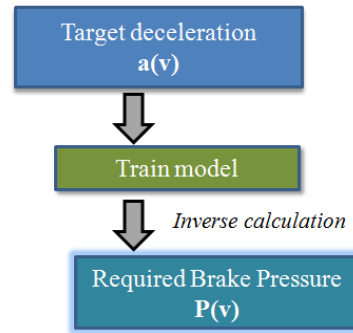


Figure 3 : Target pressure computation

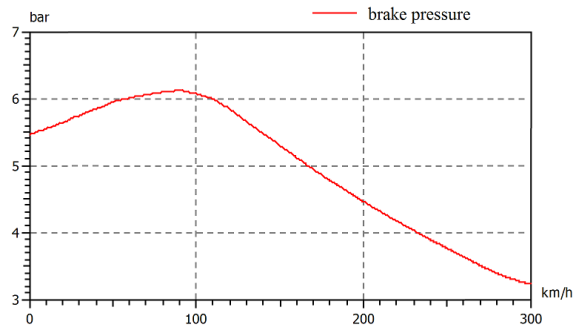


Figure 4 : Computed target brake pressure

The possibility to do inverse computation makes this sizing step smoother and easier to handle.

3 Components & Valves modeling

Once a target pressure is defined as a function of speed and deceleration, detailed design can start. We introduce in this part models that are used by valves designers.

3.1 Pneumatic model - generalities

The gas properties for air are considered as ideal gas, this model holds since the system works at pressure below 10 bar, around room temperature.

Volumes are lumped volumes using mass and energy balance to compute the pressure and temperature derivatives in a pretty much conventional fashion.

Flow models for orifices are based on geometrical flow area and a flow coefficient. Sonic flow is accounted for when a given critical pressure ratio is reached.

3.2 Valve models

We will only detail the Electro-Pneumatic Valve (EPV) model, other models being similar. We intro-

duce the basic working principle of the EPV in Figure 5.

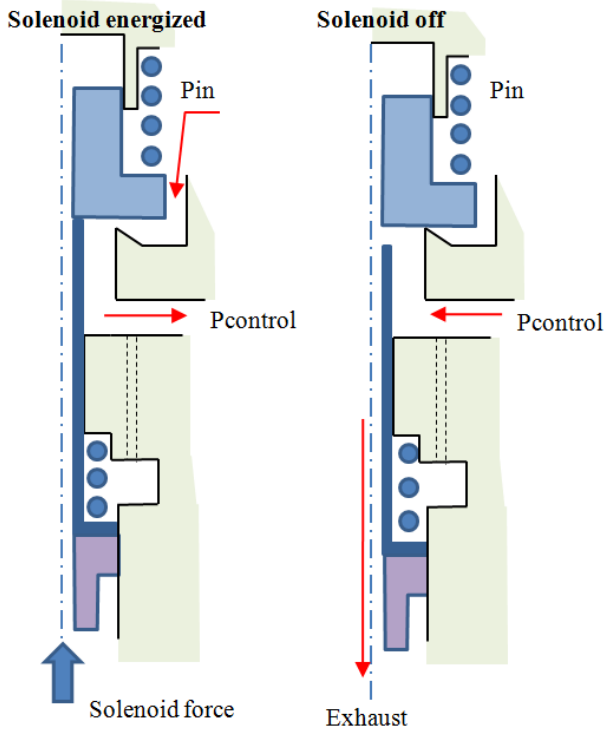


Figure 5 : EPV working principle

The solenoid current is used to adjust the control pressure applied in the brake calipers. The modeling of the solenoid is discussed in the next part.

The model considers 2 moving bodies inside the valve. For each moving body the forces of springs and of pressure areas are taken into account. Sealing friction can also be accounted for but can be usually neglected.

The relative position of each bodies determines the opening of the flow areas of the valve, connecting the inlet pressure to the control pressure or control pressure to exhaust. The variable flow area is defined by an expression for the orifice area as a function of the bodies position. This is achieved through an orifice with time dependent parameter for its area. The time dependent parameters in *SimulationX*[®] is quite a convenient feature because of the great flexibility it gives to any component. For a flapper nozzle valve, as found in the EPV, the flow area can be written as :

$$A(x) = \pi D x,$$

where D is the seat diameter.

The resulting model structure is provided in Figure 6. The solenoid model is discussed in the next section.

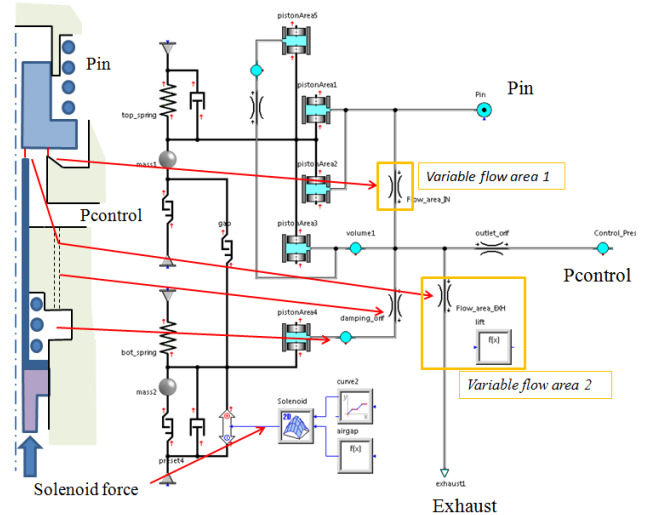


Figure 6 : EVP model structure

For pneumatic valves, it is also common to find designs with membranes. Membranes are modeled as pistons with variable effective area for the pressure force. The effective area is computed from the membrane volume variation by the equation :

$$\frac{dV(x)}{dx} = A_{\text{eff}}(x),$$

Where V(x) is the membrane volume and A_{eff} is the effective area of the piston.

We illustrate, in Figure 7, a key output for the EPV model : the curve giving the control pressure as a function of the current. The curve is computed with a current ramp starting at 0 and going down to 0, we can notice that the hysteresis of the output pressure is predicted.

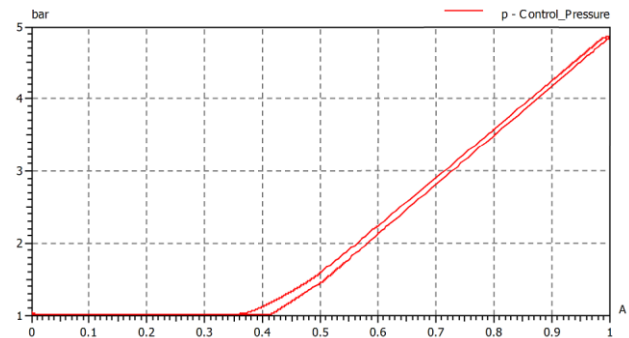


Figure 7 : EPV characteristic curve (Current/Pressure)

The detailed valve model presented helps designers define flow areas, geometries, springs properties, assess valve stability, and assess flow performance along with pressure regulation performance. These models are a key tool to achieve successful designs of such valves.

3.3 Solenoid model

The Solenoid is modeled by the use of a 2D table giving the force as a function of airgap and current:

$$F_{\text{solenoid}}[\text{N}] = f(\text{airgap}[\text{m}], \text{current} [\text{A}])$$

The table is directly implemented into the model described in Figure 8. The data to feed the table can be obtained from experiments or from FEA analysis. A more detailed model, for design purpose of the solenoid, can be done using 1D lumped magnetic element and FEM analysis for the reluctance of airgaps, as shown in Figure 9. It is also possible to include thermal simulation in this part and verify that the solenoid temperature remains in acceptable ranges.

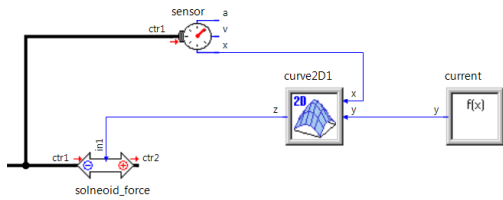


Figure 8 : EPV Solenoid model in *SimulationX*

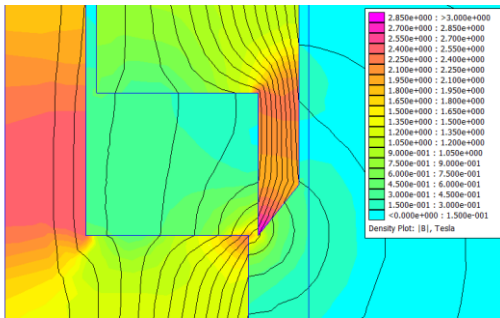


Figure 9 : Detail of a FEM analysis for a solenoid airgap, flux density and flux lines

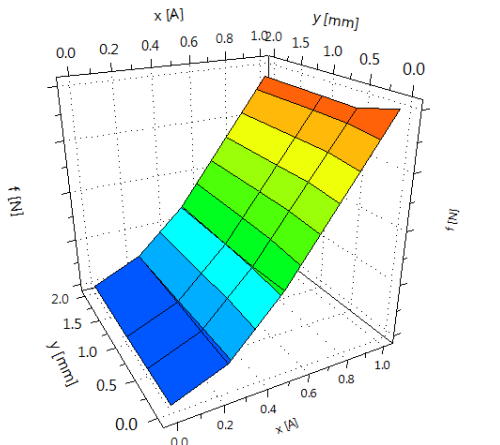


Figure 10 : Solenoid force, as a function of airgap [mm] and current [N]

3.4 Calipers and brake model

The calipers mechanical system consists of brake cylinder connected to the brake pads through a lever system, as shown in Figure 12. The corresponding model is shown in Figure 12.

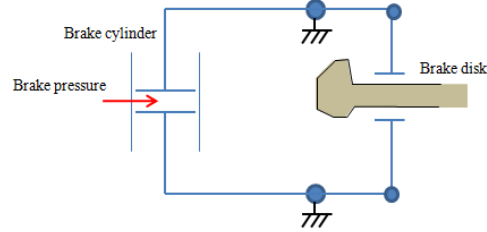


Figure 11 : Schematic of brake caliper

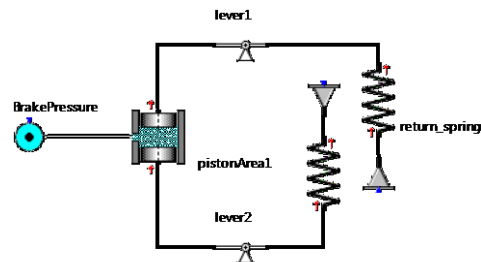


Figure 12 : brake cylinder – brake caliper mechanism
For the brake pad contact and brake torque, we use the following equation:

$$T [\text{N.m}] = R[\text{m}] \lambda_{\text{brake}}(v) F_{\text{brake}}[\text{N}],$$

where T is the brake torque, R the mean application radius, $\lambda_{\text{brake}}(v)$ the friction coefficient between the brake pad and the disk, as a function of the relative speed and F_{brake} the normal applied on the brake pad and disk. The variable friction coefficient as a function of speed needs also to be taken into account in the initial sizing step (part 2 of this paper) of the braking system, since it will modify the required target pressure.

4 Wheel-rail interface

The wheel-rail interface is an important part for simulating the Wheel Slide Protection device (WSP). The role of the WSP is to prevent wheel slide under all conditions. The properties of the rail contact have a major impact on the wheel adherence and on the WSP behavior. For this reason, we need to create a model of the wheel-rail interface.

An important output of the wheel-rail contact model is the creep relationship with creep force. Creep is defined as the relative slip between the wheel and the rail, creep force is the resulting force opposed to the

direction of motion. Typical curves are provided in Figure 15.

4.1 Summary of the contact theory

We introduce in Figure 13 the overall geometry of the rail-wheel contact problem. The Hertz theory is applied and provides a solution for the contact patch between rail and wheel. The contact patch is an ellipse of semi-axes a and b , as shown in Figure 14.

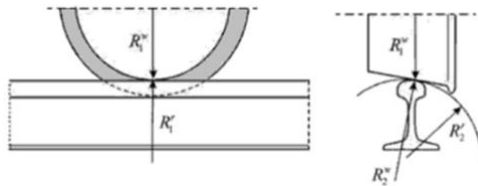


Figure 13 : Wheel-rail geometry

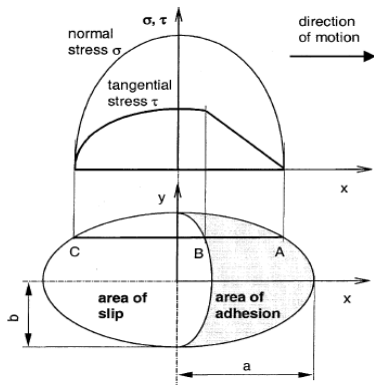


Figure 14 : contact patch geometry

The Hertz theory gives also a formula giving a relation for the semi-axes a and b :

$$a = f(F_n, E_i, \nu_i, R_i),$$

where F_n [N] is the normal force, E_i [Pa] the Young modulus of each body, ν_i the Poisson ratios and R_i the curvature radius at the contact point. The tangential force F_x is computed according to the model proposed in [9] which applies Kalker's linear theory and provides :

$$F_x = \frac{2 F_y \mu}{\pi} \left(\frac{k_a \epsilon}{1 + (k_a \epsilon)^2} + \arctan(k_s \epsilon) \right)$$

$$\epsilon = \frac{G \pi a b}{4 F_y \mu} C_{11},$$

where μ is a variable friction coefficient computed as :

$$\mu = \mu_0 [(1 - A) e^{-B\omega} + A],$$

with μ_0 , A , B , k_a , k_s being parameters of the model and ω being the total creep velocity between rail and wheel.

The proposed creep force model has only 5 parameters that need to be identified on measurement. This model have the advantage of being able to cover accurately small creep and large creep conditions, while being able to account for train velocity and different rail-wheel interface conditions (ice, rain, leaves, dry...).

The main output of interest of the model is the creep force curve, as shown in Figure 15.

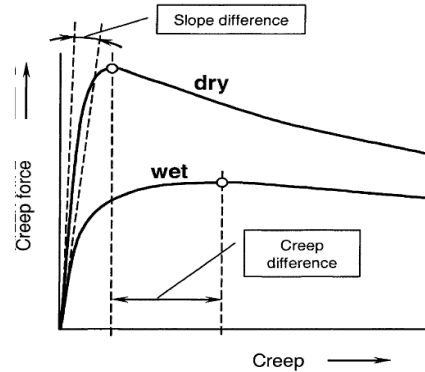


Figure 15 : Classical creep force / creep curves

4.2 Model implementation in Modelica

The model proposed described in 4.1 is implemented into Modelica through the help of the TypeDesigner in *SimulationX*[®]. The model extends a rotary inertia element and represents a wheelset accounting for the wheel-rail interface. We introduce the resulting icon with 1 rotary mechanical connection in Figure 16.

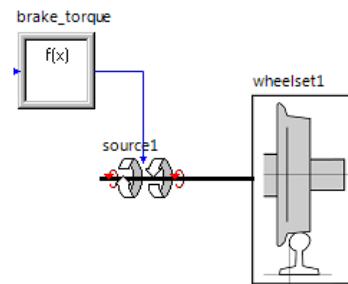


Figure 16 : Wheelset model in *SimulationX*[®].

4.3 Validation of rail-wheel contact model

Implementation validation and parameters identification of the wheel-rail contact are done using measurements from [9] , [10], [12], [13], [14].

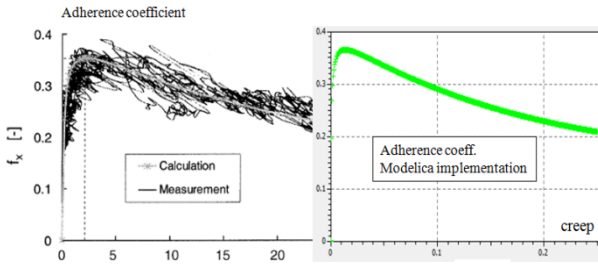


Figure 17 : Comparison of results from [9] with the modelica implementation, *Siemens locomotive S252 (dry, v=30km/h)*

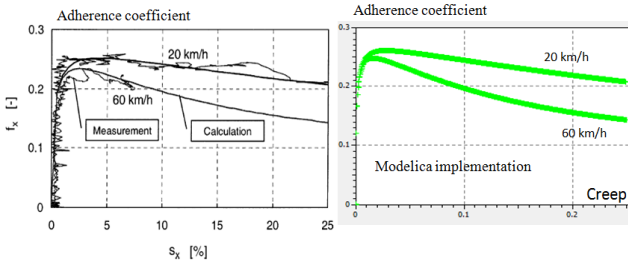


Figure 18 : Comparison of results from [9] with the modelica implementation, *Bombardier locomotive 12X (wet, V=20 and 60 km/h)*

We provide also an example of creep curves for 2 different cars under heavy rain. The front car has a reduced adherence coefficient compared to car in the middle of the train. We illustrate this effect in Figure 19.

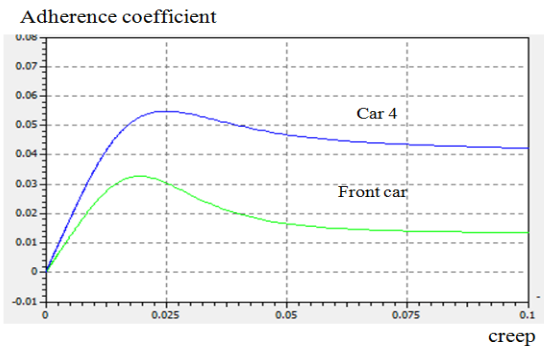


Figure 19 : adherence coefficient at 300km/h under heavy rain, comparison of front car and car 4.

The maximum adherence coefficients provided in Figure 19 are consistent with the measurement on a Japanese Shinkansen from [10], which gives values around 0.02 for the front car and 0.05 for the car 4.

Overall, parameters for the following conditions were identified :

- Dry – high adherence
- Dry – medium adherence
- Dry – contaminated surface
- Water/rain (multiple cars)
- Oil film
- Fallen leaves residues

The model being parametric, the user can also provide his values for the 5 relevant parameters.

Integrating the wheel-rail contact inside our pneumatic and mechanical model has several advantages over using a full-blown 3D multi-body specialized rail dynamics simulation package. The first advantage is that there is no need for tool couplings or model import/export. The other advantage is that we obtain a very high performance in terms of simulation time, with the possibility to easily integrate our complete model into a HIL simulator.

5 Wheel Skid Protection device

With the availability of a predictive wheel-rail contact model, it is possible to perform design and analysis of the Wheel Skid Protection (WSP) device. The device consists of a set of valves piloted by an electronic controller. These valves modulates pressure in order to maintain brake torque but without wheel blocking. Each wheelset is equipped with a WSP, requiring 2 valves by wheelset. One valve is used to close pressure input port, the second valve is used for venting the cylinder to exhaust if necessary. The valves are controlled by the BCU control logic. We show the WSP valve block model in Figure 20.

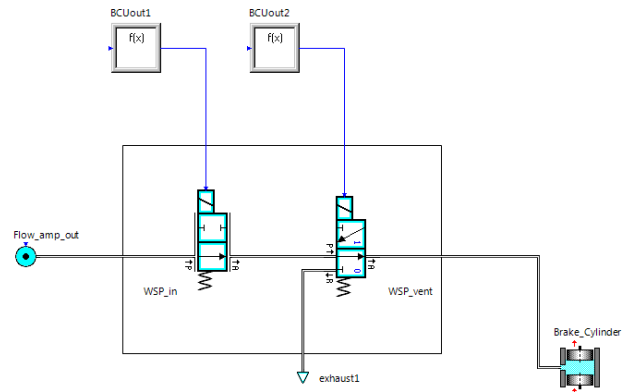


Figure 20 : WSP valve block model structure
We introduce some example of WSP action under heavy rain in Figure 21.

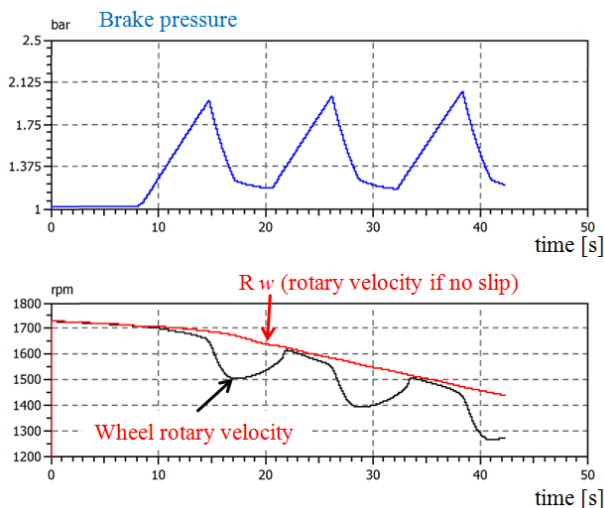


Figure 21 : WSP in action at 300km/h under heavy rain, modulation of brake pressure

The pneumatic model coupled with the wheelset model is used to assess the robustness of the WSP device and optimize the control algorithm, balancing wheel-slide protection and braking distance. The models introduced in this section provide mechanical and control designers with a valuable simulation tool for achieving robust and reliable performance.

6 Rolling-stock

The rolling-stock is considered as one mass, or several masses linked together through non-linear spring-dampers. Rolling-stock models needs to account for aerodynamic friction, rolling friction and dry friction. These are implemented as additional forces depending on train velocity.

The system we consider in this article does not use an air-spring load sensing system. If this was the case, a simplified 3D mechanical model of the rolling-stock could be created in order to assess the air-spring pressure during braking.

7 System modeling and HIL simulation

7.1 System model on the laptop

The models developed previously are integrated into a system model :

- Compressors and feed lines
- pneumatic tanks, main valves, pipes
- WSP valves
- brakes model
- wheelset with wheel rail contact

- rolling-stock (car)

The system model on the laptop can be used for a MIL (Model In the Loop) step to design the control strategy. It can be used to assess the performance of the Wheel Slide Protection (WSP) device before an actual hardware implementation. The WSP device should insure that the wheel cannot be blocked during braking by modulating the caliper pressure but also maintaining a short braking distance as stipulated by safety regulations (<3000m at 250 km/h for instance).

Multiple configurations can be evaluated and tested very efficiently, including fault and failure simulations on a complete train composed of 8 to 16 cars.

7.2 Real time model

Besides the need for laptop simulation, it is also possible to use the models to test the controller hardware in a HIL simulator. To achieve this, we need real-time capable models, which will be different than the detailed design models.

The real-time models for components and valves are developed and integrated into a real-time system level model. The model is tested off-line with a fixed step solver and its accuracy is compared with the detailed model developed previously. Numerical stability of the solution is achieved around $1e-3$ [s] time step, while maintaining a safe margin for CPU time compared to real-time.

7.3 C-code export and HIL simulator

The model can then be exported for the HIL simulator that is used to develop and test the Brake Control Unit. *SimulationX* c-code export is used. The c-code can be either integrated into a s-function for Simulink and used then for exporting with RTW, but it is also possible to directly export the model for a given real-time target such as a dSpace system, a NI Veristand system or a Cosateq Scale-RT based system. Depending on the cases and requirements, it is possible to interface the input/output cards of the HIL simulator directly inside the *SimulationX* model, using a I/O boards library.

8 Conclusions

A methodology for developing and testing high-speed train pneumatic braking system has been demonstrated. The existing modelica tools have been

extended with a wheel-rail contact model in order to simulate the wheel-slide protection device.

The proposed tool can then cover needs from sizing and requirements definition, detailed component design down to HIL simulation for control validation and testing.

References

- [1] *Analysis of the Braking System of the Korean High-Speed Train Using Real-time Simulations*, Chul-Goo Kang, *Journal of Mechanical Science and Technology* 21, 1048-1057, 2008
- [2] *Method of Analysis for Determining the Coupler Forces and Longitudinal Motion of a Long Freight Train in Over-the-Road Operation*, G. C. Martin, W. W. Hay, *Civil Engineering Studies, transportation Series No.2*, June 1967
- [3] *Braking Systems - Railway Technology Today* 7, Izumi Hasegawa, Seigo Uchida, *Japan Railway & Transport Review* 20, June 1999
- [4] *Modeling the longitudinal dynamics of long freight trains during the braking phase*, Luca Pugi, Duccio Fioravanti, Andrea Rindi, *12th IFToMM World Congress, Besancon (France)*, June18-21, 2007
- [5] *Investigation of the dynamics of railway bogies subjected to traction / braking torque*, Yunendar Aryo Handoko, *Centre for Railway Engineering, Central Queensland University, Australia*, Sept. 2006
- [6] *A Fast Wheel-Rail Forces Calculation Computer Code*, Oldrich Polach
- [7] *Railroad vehicle dynamics: a computational approach*, Ahmed A. Shabana, Khaled E. Zaazaa, Hiroyuki Sugiyama, *CRC Press*, 2008
- [8] *Modelling and model validation of heavy-haul trains equipped with electronically controlled pneumatic brake systems*, M.Chou, X. Xia, C. Kayser, *Control Engineering Practice* 15, 2007
- [9] *Creep forces in simulations of traction vehicles running on adhesion limit*, O. Polach, *Wear* 258, 2005
- [10] *Measurement and analysis of adhesion phenomena in high speed train*, Masanobu Nankyo, Shin-ichi Nakazawa, *Proceedings of IMECE2008, IMECE2008-66455*, 2008
- [11] *Brake Technology Handbook*, Bert Breuer, Karlheinz H. Bill, 2006
- [12] *Adherence en freinage et anti-enrayeurs, document technique DT257*, M.Boiteux, M.Cadier, J.King, W. Kunes, *office de recherche et d'essai de l'Union International des Chemins de fer*, 1992.
- [13] *Effect of oil and water mixtures on adhesion in the wheel/rail contact*, R. Lewis, E A Gallardo-Hernandez, *Proc. ImechE vol 223 Part F*, p 275
- [14] *The "leaves on the line" problem – a study of leaf residue film formation and lubricity under laboratory test conditions*, P.M. Cann, *Tribology letters*, Vol. 24, No2, Nov 2006

An Advanced Environment for Hybrid Modeling and Parameter Identification of Biological Systems

Sabrina Proß Bernhard Bachmann
 University of Applied Sciences Bielefeld
 Am Stadtholz 24, 33609 Bielefeld, Germany
<http://www.fh-bielefeld.de/ammo>

Abstract

Biological systems are often very complex so that an appropriate formalism is needed for modeling their behavior. Hybrid Petri nets, consisting of time-discrete as well as continuous Petri net elements, have proven to be ideal. This formalism was implemented based on the Modelica language. Several Petri net components are structured within an advanced Petri net library. A special sub-library contains so-called wrappers for specific biological reactions to simplify the modeling procedure.

The Petri net models developed with the Dymola tool can be connected to Matlab Simulink to use all the Matlab power for parameter identification, sensitivity analysis and stochastic simulation.

This paper illustrates the usage of the Petri net library, the coupling to Matlab Simulink and further processing of the simulation results with algorithms in Matlab. In addition, the application is demonstrated by modeling the metabolism of Chinese Hamster Ovary Cells.

Keywords: Biological Systems; Petri nets; Parameter Identification

1 Introduction

The procedure of modeling and simulation is displayed in Figure 1. The beginning of the process is founded by a set of objectives and purposes which are translated together with current knowledge of the system into a list of specific hypotheses.

The next stage is to find a formalism which can model the defined hypotheses by specific mathematical equations. The modeling of biological systems demand often a combination of continuous and discrete equations, a differential equation system sole is often not sufficient. Examples are gene regulation and processes where the organism switches from substance production to consumption or vice versa when special environmental conditions occur. This

kind of process takes place within the metabolism of Chinese Hamster Ovary Cells (CHO-Cells) which switches from lactate and ammonium production to consumption after a specific change of environmental conditions. In addition, the antibody production starts only when special environmental conditions occur. The concrete mechanism is part of section 5.

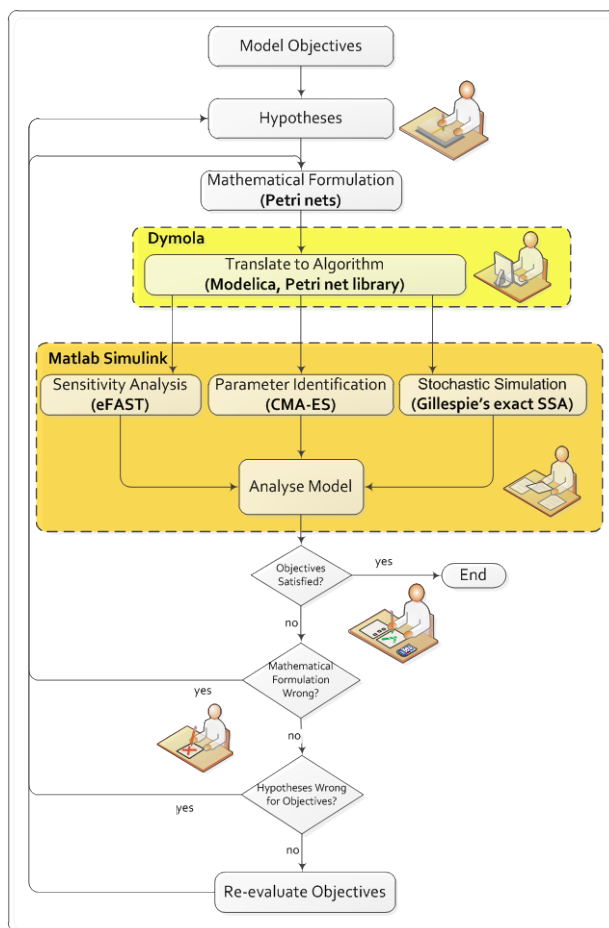


Figure 1: The modeling procedure

The hybrid Petri net concept, consisting time-discrete as well as continuous Petri net elements, fulfills all the required biological conditions and is thusly applicable for the realization of the CHO-model. The biological pools, like e.g. metabolites, genes, proteomes and signals are represented by

places. The reactions between them can be modeled by transitions. This transfer of biological systems to Petri nets was first introduced by Reddy [1]. An introduction in the basic Petri net concepts is given in section 2.

Since the Modelica language provides all necessary features to implement the Petri net formalism it has been chosen to develop an advanced Petri net library, whereby the Petri net component models consist of differential, algebraic and discrete equations. Section 3 gives an introduction to the Petri net library.

On the basis of an established Petri net model, several calibration and analysis methods can be applied by coupling Dymola with Matlab Simulink. Section 4 gives an introduction in selected methods for parameter identification, sensitivity analysis and stochastic simulation as well as their realization within Matlab Simulink.

The modeling procedure is done if the model satisfies all determined objectives. Otherwise the mathematical formulation is wrong or the hypotheses are not correct for the desired objectives. In this case the modeling procedure has to be restarted at the respective stage.

2 Petri Nets

The Petri net formalism for graphical modeling of concurrent and nondeterministic processes was first introduced by Carl Adam Petri in 1962 [2]. A Petri net is mathematically a directed, 2-colored and bipartite graph. The property 2-colored implies the division in two unique node sets which are called *transitions* and *places* and only places can be connected to transitions or transitions to places according to the bipartite attribute. The places are represented graphically by circles and transitions by rectangles. A place models a state, for example of an object or a condition, while a transition models the change of states, for example activities or events.

Every place can contain an integer number of *tokens*. These tokens are represented graphically by little, black dots or numbers inside the places. A concrete determination of the token number of a place is called *state of the place* and a concrete determination of the token numbers of every place is called the *state of the Petri net*. Furthermore, the directed edges can have integer weightings which are written at the edges.

Following all places in the previous area of a transition are called *previous places* and all in the past area are called *past places*. Similarly, the transitions in the previous area of a place are called *previous*

transitions and all in the past area are called *past transitions* (see Figure 2).

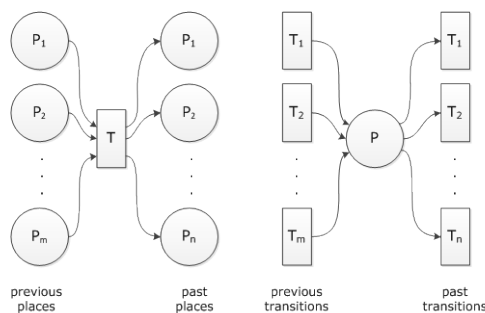


Figure 2: Previous and past places, and previous and past transitions

A transition is *ready-to-fire* if all previous places have at least as much tokens as the edge weightings. A ready-to-fire transition *fires* by removing as much tokens as the edge weightings from all previous places and by adding as much tokens as the edge weightings to all past places.

Figure 3 shows at the top an example of a Petri net where the transitions $T1$ and $T2$ are ready-to-fire and the others not. The Petri net at the bottom displays the new state after firing transition $T1$ and $T2$.

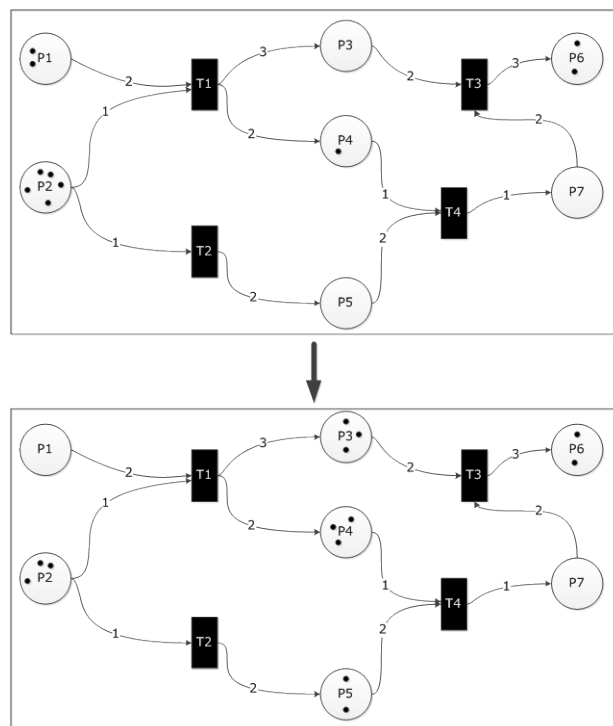


Figure 3: Petri net example, top: transitions $T1$ and $T2$ are ready-to-fire, bottom: new state of the Petri net after firing transition $T1$ and $T2$

In the last years, the basic Petri net concept, described above has been more and more extended in order to model different kind of applications (e.g. biological systems). The first extension is that every place in a Petri net has a lower and upper limit of

tokens. These Petri nets are called *Petri nets with capacities*.

Biological applications demand not alone capacitated places but also limitations especial for each edge from places to transitions. In this conjunction, the lower bound of an edge is called *threshold*, the upper bound is called *inhibition* and the Petri net is called *Petri net with edge bounds*.

The fixed edge weightings can be replaced by dynamical ones which may depend on the current token number of a place. In this manner, not only integers can be written at the edges but also the name of a place. This Petri net extension is called *self-modified Petri net* and was first introduced by Valk [3].

These self-modified Petri nets can be further expanded to *functional Petri nets* by allowing functions as edge weightings which may depend on token numbers of several places [4].

For the simulation of a Petri net it is necessary to associate time with its behavior. One possibility to do this is that every transition gets a delay. A delay is the time period that the respective state change takes. This Petri net concept is called *timed Petri net*.

This concept can be modified to *stochastic Petri nets* by random delays, i.e. the fixed values are replaced by random numbers that change at every activation point in time. The delays are exponentially distributed random numbers, whereby the characteristic parameter λ can depend on token numbers of several places (see e.g. [5], [6]).

Biochemical reactions occur in most cases continuously. In order to model these reactions, the discrete Petri net concept has to be transferred to a continuous one [7]. The most serious difference between discrete and *continuous Petri nets* is that token numbers are real and that transitions fire continuously. A function is assigned to every edge of a continuous Petri net depending on token numbers of several places just like functional Petri nets. These functions specify the speed of the firing process and are the right side of differential equations. A continuous Petri net is an ordinary differential equation system whose structure can change within time.

Additionally, the modeling of biological systems demands often a combination of discrete and continuous processes. Hybrid Petri nets which contain discrete as well as continuous Petri net elements accomplish this [8].

The following connections are allowed within hybrid Petri nets

- ✓ discrete place → discrete transition
- ✓ discrete transition → discrete place
- ✓ continuous place → continuous transition

- ✓ continuous transition → continuous place
 - ✓ continuous place → discrete transition
 - ✓ discrete transition → continuous place
- Not allowed are the connections
- ✗ discrete place → continuous transition
 - ✗ continuous transition → discrete place

3 Petri Net Library

The advanced Petri net library described in this paper bases on the previous ones developed in Modelica ([9], [10], [11]). The improvements are:

- Discrete Petri nets
 - Edges can have integer or functional weightings depending on token numbers of several places
 - Edges can have integer bounds (threshold and inhibition values)
 - If a place has a bottleneck, the connected transitions are enabled randomly with different probabilities
- Continuous Petri nets
 - Generalization of the discrete Petri net concept to the continuous one
 - Edges can have functional weightings depending on token numbers of several places
 - Places can have minimum and maximum capacities and edges can have bounds (threshold and inhibition values)
- Hybrid Petri nets
 - Combination of discrete and continuous Petri net elements to hybrid Petri nets

The advanced Petri net library is structured in five sub-libraries: Discrete, Continuous, Stochastic, Reactions and Global. Additionally, there are packages for Interfaces, Constants, Functions and Blocks which are used within component models (see Figure 4). The Petri net elements of the library are represented by the icons in Figure 5.

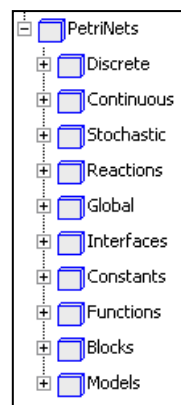
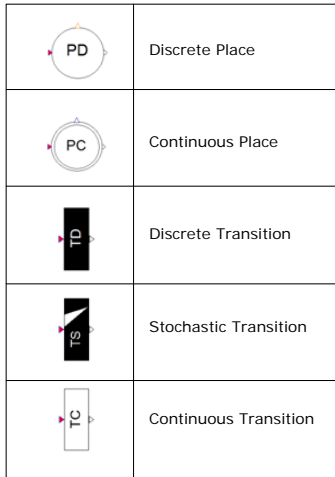


Figure 4: Structure of the Petri Net library


Figure 5: Icons of the Petri net library

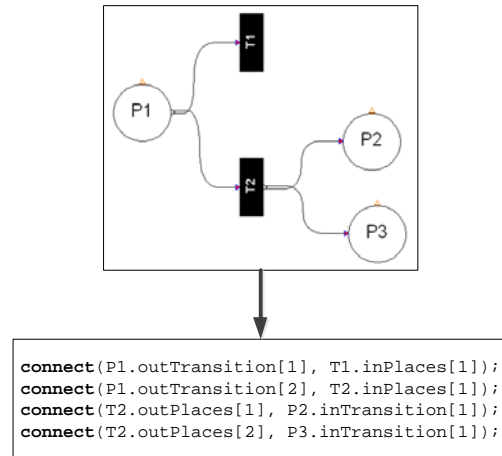
The implementation details of places and transitions can be found in [12]. This Petri net library has been improved concerning the connectors of places and transitions since new modeling features are available in Modelica 3.2. The components have been upgraded by using the `connectorSizing` annotation [13].

```
parameter Real nPast=1 annotation
  (Dialog(connectorSizing=true);
parameter Real nPre=1 annotation
  (Dialog(connectorSizing=true);
```

The parameters `nPast` and `nPre` are used as dimension size of the vectors of connectors. The Dymola tool set these parameters automatically, i.e. they appear not in the property dialog. If a new connection is drawn, the respective parameters are incremented by one and a new `connect`-equation is created for the new highest index. Figure 6 shows a Petri net example as component diagram and the corresponding `connect`-equations that are created automatically by drawing a line from place to transition. In regard to this upgrade, some of the parameters of the place and transition models have to be entered as vectors since they belong to their edges. Exemplary, the vector parameter `add` of a transition which contains all weightings of the edges to its past places (cp section 3.2). The weighting of the edge from transition `T2` in Figure 6 to place `P2` is supposed to be 5 and the weighting of the edge from `T2` to `P3` is supposed to be 8. Then the parameter `add` has to be `add={5, 7}`. The first entry in the `add`-vector corresponds to the first connection starting from `T2` indexed with [1] and the second entry corresponds to the second connection indexed with [2].

The drawback of this concept is that the knowledge about the indices of the connections is needed to guarantee the right assignment of the edge weightings. When a connection is added or deleted one

must keep attention that the entries of the vector parameters are still in the right order. But on the other hand the big advantage is that components can have an arbitrary amount of previous and past, which simplifies the modeling process enormously.


Figure 6: The `connectorSizing` annotation

3.1 Place Model

Table 1 contains the parameters which can be set in all places (discrete, stochastic and continuous) and Table 2 shows those that are only part of discrete places. The parameter `enablePast` is explained in Figure 7.

Table 1: Parameters of both places (discrete and continuous)

Identifier	Description	Type/Default
startTokens	The number of tokens that the place contains at the beginning of the simulation. In the discrete case nonnegative integer and in the continuous case nonnegative real numbers can be entered.	scalar/ zero
minTokens	The minimum number of tokens that the place must always contain. In the discrete case nonnegative integer and in the continuous case nonnegative real numbers can be entered.	scalar/ zero
maxTokens	The maximum number of tokens that the place can contain. In the discrete case nonnegative integer and in the continuous case nonnegative real numbers can be entered.	scalar/ infinite

Table 2: Parameters only of discrete places

Identifier	Description	Type/Default
enablePast	Enabling probabilities of the past transitions. If a place has not enough tokens to enable all its past transitions, a random decision must be made whereby the respective transition is chosen with the entered probability. The sum of all enabling probabilities has to be one (see Figure 7).	vector/ each entry $\frac{1}{nPast}$
enablePre	Enabling probabilities of the previous transitions. If a place cannot gain tokens from all its previous transitions due to its maximum capacity, a random decision must be made whereby the respective transition is chosen with the entered probability. The sum of all enabling probabilities has to be one.	vector/ each entry $\frac{1}{nPre}$

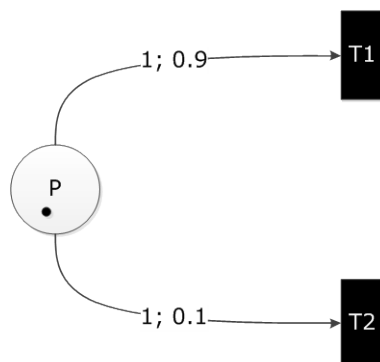


Figure 7: Petri net example for the enablePast-parameter: The place P has one token, not enough to fire in T1 and T2 simultaneously since both edge weightings are one. A random decision is applied where T1 is chosen with the probability 0.9 and T2 with the probability 0.1. The sum of all enabling probabilities of a place has to equal one.

3.2 Transition Model

The parameters available in all transition models are summarized in Table 3. Table 4 contains the parameters, which are only part of the discrete transition. Those that can be only set in stochastic transitions are shown in Table 5.

Table 3: Parameters of all transitions (discrete, stochastic and continuous)

Identifier	Description	Type/Default
sub	Weightings of edges start-	vector/

	ing from previous places. In the discrete and stochastic case nonnegative integers and functions can be entered and in the continuous one nonnegative real numbers and functions are allowed. With the “.t”-notation one can access the tokens of a place for the edge weightings, e.g. $sub = \{ 2.9 * P1.t \}$.	each entry 1
add	Weightings of edges ending in past places. In the discrete and stochastic case nonnegative integers and functions can be entered and in the continuous one nonnegative real numbers and functions are allowed. With the “.t”-notation one can access tokens of a place for edge weightings, e.g. $add = \{ 0.45 * P1.t \}$.	vector/ each entry 1
inhibition	Upper bound of edges starting from previous places. Nonnegative integers can be entered in the discrete and stochastic case and nonnegative real numbers in the continuous case.	vector/ each entry infinite
threshold	Lower bound of edges starting from previous places. Nonnegative integer can be entered in the discrete and stochastic case and nonnegative real numbers in the continuous case.	vector/ each entry zero
con	Condition which has to be true so that the transition can become active and can fire, e.g. $time > 9.7$.	scalar/ true

Table 4: Parameter only of a discrete transition

Identifier	Description	Type/Default
delay	The time that a discrete transition waits after its activation before it fires.	scalar/ 1

Table 5: Parameters only of a stochastic transition

Identifier	Description	Type/Default
c	Constant for the pre-defined lambda functions (see parameter lambdaFunc)	scalar/ 1
lambdaFunc	Pre-defined function for the lambda calculation; choice between Stochastic	scalar/ Stochastic mass

	tic mass action hazard function and Stochastic level hazard function (see [14])	action
lambda	User-defined function for lambda instead of the pre-defined lambda functions (Stochastic mass action hazard function and Stochastic level hazard function)	scalar

3.3 Reactions Sub-Library

The Petri net models of the Discrete, Continuous and Stochastic sub-libraries can be wrapped into models for different kinds of biological reactions to simplify the modeling process. These model components are organized in the sub-library Reactions which is also divided in several sub-libraries for different reaction types. Till now there are:

- o Reaction kinetics
- o Enzyme kinetics
- o Growth kinetics
- o Culture strategies
- o Process activations

The detailed wrapping process is explained in [15].

3.4 Animation in Dymola

The simulation results can be displayed by either plots of selected token numbers or by an animation. By the latter the degree of redness changes during time according to the token number of the place, i.e. a red place has many tokens and a white place is empty. The redness degree can be scaled from 0 to 100 by the green Settings-box which is a component of the Global sub-library (see Figure 8).

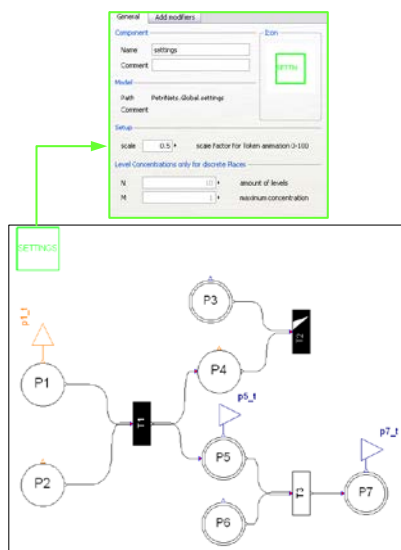


Figure 8: Scaling of the redness degree by the Settings-component of the global sub-library

This animation is realized by the annotation DynamicSelect [13]

```

annotation(fillColor = DynamicSelect
({255,255,255},if scale<100 then
{255,255-2.55*tokenscale,255-
2.55*scale} else {255,0,0})

```

Figure 9 shows the redness change of a Petri Net example during time. This animation offers a good way for analyzing large and complex Petri Nets.

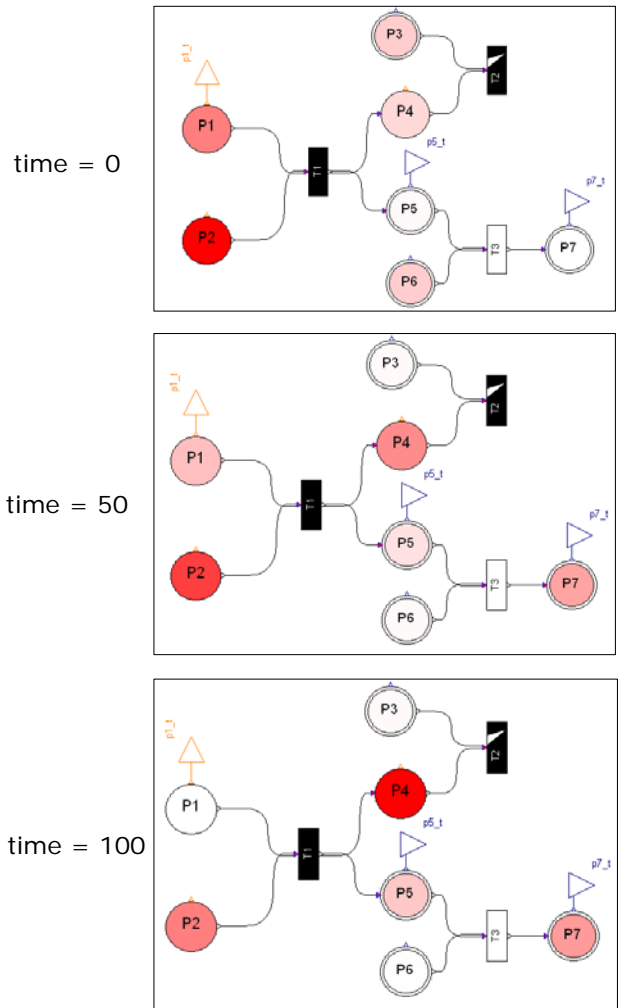


Figure 9: Animation of a Petri net in Dymola; the token distribution of the Petri net example, top: at the beginning of the simulation, middle: after a simulation of 50 time units, bottom: after a simulation of 100 time units; the degree of redness corresponds to the token numbers, i.e. a red place has many tokens and a white place is empty

4 Model Calibration and Analysis

Once a Petri net model is established, the next step according to Figure 1 is to get further insight in the model parameter characteristics by sensitivity analysis, parameter identification and stochastic simula-

tion. For this, a connection between the Petri net model in Dymola and Matlab Simulink has to be created to benefit from the power of Matlab.

4.1 Connection Dymola and Matlab Simulink

For parameter optimization, sensitivity analysis and stochastic simulation, it is necessary to simulate the model several times with different parameter settings. Dymola offers a possibility to connect a Modelica model to Matlab by a Simulink interface (DymolaBlock) and a set of Matlab m-files [16]. Figure 10 displays at the top a Petri net modeled by the Petri net library in Dymola and at the bottom the corresponding Simulink model. If the token number of a place over the time is needed in Matlab for further calculations, one has to create a connector above the respective place. This is an orange IntegerOutput connector in the case of a discrete place or a blue RealOutput connector if it is a continuous place. In the Petri Net example of Figure 10 the token numbers of the places $P1$, $P5$ and $P7$ are needed in Matlab, whereby $P1$ is a discrete place with an IntegerOutput connector and $P5$ and $P7$ are continuous with a RealOutput connector. The DymolaBlock in Simulink generates a connector for all places connected with an output connector in Dymola. These connectors can then be connected via a bus to an outport so that these simulation results are saved in a matrix and are available in the Matlab environment for further calculations. In the same manner it is also possible that Petri net models get inputs from Matlab via a connection between a Simulink source and a Modelica IntegerInput or RealInput connector.

To connect a Dymola-model with Simulink, one has to enter the model name and its path in the property dialog of the DymolaBlock (see Figure 11). After that, the model can be compiled and the parameters can be set. The parameters can be also set within Matlab by special m-files.

```
[p,x0,pnames,x0names]=loadsdin(
    'example.txt');
p=setParameterByName(pnames,p,
    'param1',25);
setParametersFDsin('ex/example1',
    pnames,p,x0names,x0)
```

For a detailed description see [16]. After the parameter setting the Simulink model can be simulated by the prompt

```
sim(model,timespan,options,ut).
```

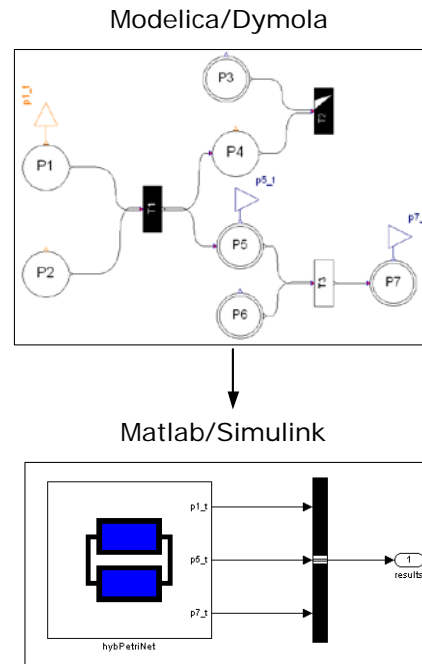


Figure 10: Connecting Dymola and Matlab Simulink by a Simulink Interface (DymolaBlock), top: Petri net modeled by the Petri net library in Dymola, bottom: Simulink interface of the Dymola-model in Matlab Simulink



Figure 11: Parameter dialog of the DymolaBlock in Simulink

4.2 Sensitivity Analysis

The goal of the sensitivity analysis is to apportion the uncertainty in model output to the different sources of uncertainty in the model input (e.g. model parameter) [17]. It can uncover technical errors in the model, identify critical regions in the input space, establish priorities for research, simplify models and defend against falsifications of the analysis [18]. The techniques can be divided in local and global methods. The *local sensitivity analysis* concentrates on the local impact of the input factors on the model and is usually performed by computing

the partial derivatives of the output functions with respect to the input factors. In contrast, the *global sensitivity analysis* considered the influence of the input factors according to a given range of variation and a probability density function.

The Petri net component models contain among others discrete equations (e.g. when-equation). Following, several events are detected within the simulation of a Petri net so that the function of token development over time is mostly not differentiable and not continuous. This engender that the local sensitivity analysis methods cannot be applied and one has to access the global methods. Various global sensitivity analysis methods are available (see e.g. [19]). Some of them were implemented in Matlab and can be chosen with respect to the model structure.

For the example in section 5 the *extended Fourier Amplitude Sensitivity Test* (eFAST) has been applied. The eFAST is a variance-based method, i.e. it uses the variance as indicator of the importance of the input factors. The enormous advantage of this method is its applicability independent of any assumptions about the model structure. That means it works for models with a linear as well as a non-linear relationship between input and output and it is unimportant if this relationship is monotonic or not. The Fourier Sensitivity Amplitude Test (FAST) was developed by Cukier and others in the 1970s ([20], [21], [22]) and extended by Saltelli and others in 1999 [23]. The main idea behind the FAST is to convert the m -dimensional integral of the mean value of the output Y

$$\langle Y \rangle = \int \dots \int Y(\mathbf{x}) \cdot P(\mathbf{x}) d\mathbf{x}$$

into a one-dimensional integral in s by using the transformation functions, called *search curves*

$$x_i = G_i(\sin(\omega_i s)),$$

where \mathbf{x} is the n -dimensional vector of input factors, $P(\mathbf{x}) = \prod_{i=1}^m P_i(x_i)$ is the product of their probability density functions, $s \in (-\pi, \pi)$ is the *search variable* and $\{\omega_i\}$ is a set of integer angular frequencies. If the frequencies ω_i and the search curves $G_i(\sin(\omega_i s))$ are chosen appropriate, the expectation of Y can be approximated by

$$E(Y) = \frac{1}{2\pi} \int_{-\pi}^{\pi} Y(s) ds,$$

where $Y(s) = Y(G_1(\sin(\omega_1 s)), \dots, G_n(\sin(\omega_n s)))$. The variance of Y can be approximated by the Fourier coefficients A_k and B_k [24]

$$Var(Y) \approx 2 \sum_{k=1}^{\infty} A_k^2 + B_k^2,$$

where

$$A_k = \frac{1}{2\pi} \int_{-\pi}^{\pi} Y(s) \cos(js) ds$$

$$B_k = \frac{1}{2\pi} \int_{-\pi}^{\pi} Y(s) \sin(js) ds.$$

The expressions $E(Y)$ and $Var(Y)$ provides a way to estimate the expectation and the variance of the output Y . Additionally, the application of the FAST method demands a definition of the frequencies ω_i , the search curves G_i and the number of sufficient points at which the model is evaluated to allow a numerical approximation of the expectation and the variance. For this it is referred to ([20], [21], [24], [23], [19]).

The contribution of factor x_i to the variance of Y can then be approximated by the partial variance

$$D_i = 2 \sum_{p=1}^M (A_{p\omega_i}^2 + B_{p\omega_i}^2),$$

where M is the maximum harmonic that is considered [22]. The ratios

$$S_i = \frac{D_i}{Var(Y)}$$

provide a way to rank the input factors according to their contribution to the variance of output Y , called *first-order sensitivity coefficients*. An implementation of the FAST method can be found in [25].

This method has been improved by Saltelli and others [23] to the extended FAST method (eFAST) which computes the total (all-effects) contribution of each input factor to the output variance. This is done by assigning a usually high frequency ω_i to an investigated factor x_i and a set of almost identical and usually low, but different from ω_i , frequencies $\omega_{\sim i}$ to all remaining factors. The partial variance of the complementary set can be computed by

$$D_{\sim i} = 2 \sum_{p=1}^M (A_{p\omega_{\sim i}}^2 + B_{p\omega_{\sim i}}^2),$$

whereby $D_{\sim i}$ measures the effect of any orders that do not involve the factor x_i . The *total sensitivity coefficients* are then given by

$$TS_i = 1 - \frac{D_{\sim i}}{Var(Y)}.$$

For the choice of the frequencies and a detailed description of the eFAST method, it is referred to [23].

4.3 Parameter Identification

The parameter identification deals with methods that estimate the unknown model parameter to adapt the model behavior as good as possible to the reality e.g. to the measured experimental data. The objective of the optimization procedure can be for example the least squares

$$f(\mathbf{x}) = \sum_{j=1}^n \sum_{l=1}^r \left(\frac{y_j(t_l, \mathbf{x}) - d_j(t_l)}{d_j(t_l)} \right)^2 \rightarrow \min \quad \text{Eq. 1}$$

where \mathbf{x} is the input vector, $y_j(t_l, \mathbf{x})$ is the j -th model output corresponding to the input \mathbf{x} at time t_l , $d_j(t_l)$ is the measurement of the j -th output at time t_l , n is the number of measured outputs and r is the number of measured points in time. Several numerical methods are well known to solve this problem. An overview can be found in [26]. However, if the underlying model is a Petri net the standard methods like Gauss-Newton or Levenberg-Marquardt are inoperative due to the non-differentiability and non-continuity of the model output. Global optimization methods has to be use that work without derivatives. For an overview of global optimization methods it is referred to [27]. In the example in section 5 a special evolution strategy, the *Covariance Matrix Adaption Evolution Strategy* (CMA-ES), is applied to estimate the unknown model parameters. Evolution strategies consist in general of the following steps:

1. *Initialization*: a specific number of individuals is generated by a random procedure.
2. *Recombination*: one or more parents produce one or more offspring. Several methods are documented (see e.g. [28]).
3. *Mutation*: minor change of the offspring.
4. *Selection*: a specific number of the best individuals form the parents of new generation. Several methods can be found in [27]. The next iteration begins with the recombination.

The parameter estimation with the CMA-ES algorithm bases on the mentioned steps above and additionally two main principles plays an important role [29].

The first is to increase the probability of a successful mutation according to the maximum likelihood principle. Therefore, the mean of the distribution is updated such that the likelihood of previously successful candidate solutions is maximized. Furthermore, the covariance matrix of the distribution is updated such that the likelihood of previously realized successful steps to appear again is increased. These updates can be interpreted as a natural gradient descent and consequently the CMA conducts an iterated principal component analysis of successful steps while retaining all principle axes. The covariance matrix adaption is to learn about the second order model of the underlying objective function.

The second is to record two paths of time evolution of the distribution mean of the strategy. Such a path contains important information about the correlation between consecutive steps. The first path is used for

the covariance matrix adaption procedure and the second is used for a step-size control. A detailed description of the algorithm and its implementation can be found in [29].

4.4 Stochastic Simulation

Stochastic simulation is to simulate many realizations of a stochastic model (stochastic Petri net) and to study the arising results. One method is Gillespie's algorithm [30] which was created to simulate chemical and biochemical reaction systems efficiently and accurately. It is a modification of the Monte Carlo method. The elementary steps according to an underlying stochastic Petri net model are:

1. *Initialization*: Initialize the number of tokens in the stochastic Petri net, reaction constants, and the random number generators.
2. *Monte Carlo step*: Generate random number to determine the next transitions to fire as well as their delays. The delays are exponentially distributed random numbers, whereby the characteristic parameter λ is proportional to token numbers of the previous places.
3. *Update*: Update token numbers based on the firing transitions.
4. *Iterate*: Go back to step 2 unless the simulation time has been exceeded.

5 Example: Modeling the metabolism of Chinese Ovary Cells

The Chinese Hamster Ovary (CHO) Cells produce antibodies which are part of many pharmaceuticals [31]. Additionally, they release the waste-products lactate and ammonium which can inhibit their growth and antibody production when specific concentrations are exceeded ([32], [33], [34]).

Experiments were performed by growing the CHO-Cells in shaking flasks, whereby they were fed with the nutrients glucose and glutamine. They produced by conversion of these nutrients antibodies, ammonium and lactate. By the latter ones it is assumed that they cannot only be produced by the CHO-Cells but also consumed when the environmental conditions are appropriate ([35], [36]). Figure 12 displays these coherencies and Figure 13 represents the experimental data of CHO-Cells growing in shaking flasks.

The experiments were performed by the University of Applied Sciences Bielefeld, Biotechnology department [37]. The cells grow till day 4 (exponential growing phase) afterwards they pass over to the stationary phase for 2 days where approximately as

much cells grow as die. Finally, more cells die than new ones grow thus the curve of living cells decreases and the curve of death cells increases (death phase).

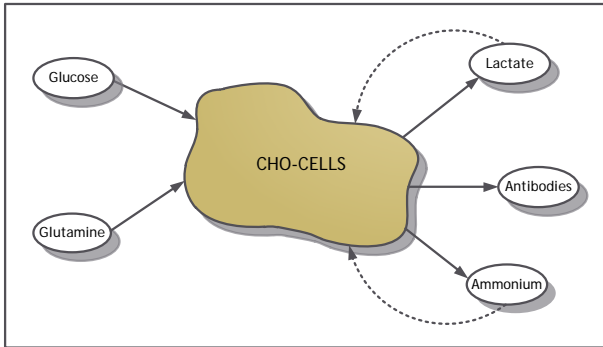


Figure 12: The main metabolism of the CHO-cells

The nutrient Glucose is exhausted at the end of the experiment and the waste-product lactate is produced till day 4 and afterwards it is consumed. They convert it back to pyruvate which enters the citric acid cycle (TCA-cycle) [36]. Here, it is assumed that they start the lactate consumption when a specific lactate concentration is exceeded. Additionally, the ammonium concentration decreases after 4 days and the glutamine concentration increases. In this conjunction, it seems likely that the CHO-cells can convert ammonium back to glutamine when the glutamine concentration falls below a specific value. The Antibody production starts first after 2.5 days and does not stop until the end of the experiment. At this point the supposition is that the cells start the production first when the glucose becomes limiting.

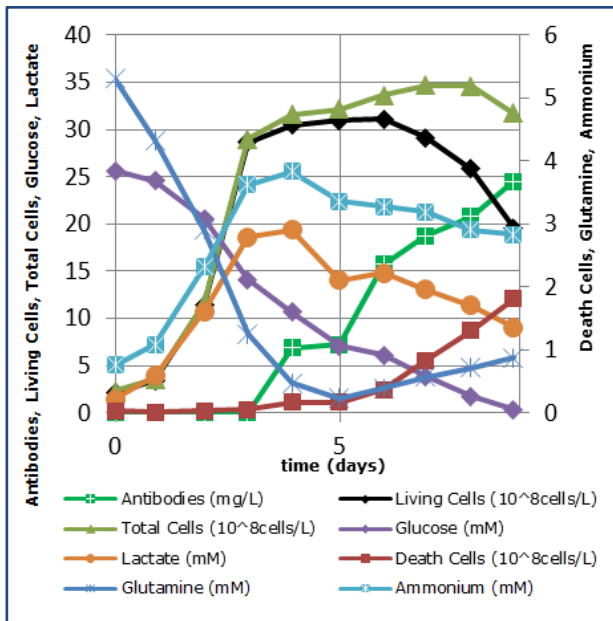


Figure 13: Experimental data of CHO-Cells growing in shaking flasks

A continuous Petri Net models the dynamics of the CHO-cells (see Figure 14). This Petri Net covers a lot of different differential equation systems. Which of them is chosen depends on the environmental conditions. At the beginning of the experiment, it represents the following ODEs

$$\frac{dX_t}{dt} = \mu \cdot X_v \quad \text{Eq. 2}$$

$$\frac{dX_d}{dt} = \mu_d \cdot X_v \quad \text{Eq. 3}$$

$$\frac{dX_v}{dt} = (\mu - \mu_d) \cdot X_v \quad \text{Eq. 4}$$

$$\frac{dGlc}{dt} = -q_{glc} \cdot X_v \quad \text{Eq. 5}$$

$$\frac{dGlu}{dt} = -q_{glu} \cdot X_v - k_{sd} \cdot Glu \quad \text{Eq. 6}$$

$$\frac{dLac}{dt} = q_{lac} \cdot X_v \quad \text{Eq. 7}$$

$$\frac{dAmm}{dt} = q_{amm} \cdot X_v + k_{sd} \cdot X_v \quad \text{Eq. 8}$$

$$\frac{dAb}{dt} = 0 \quad \text{Eq. 9}$$

$$\begin{aligned} X_t(0) &= X_{t0}, X_d(0) = X_{d0}, X_v(0) = X_{v0}, \\ Glc(0) &= Glc_0, Glu(0) = Glu_0, \\ Lac(0) &= Lac_0, Amm(0) = Amm_0, \\ Ab(0) &= Ab_0 \end{aligned} \quad \text{Eq. 10}$$

where X_t is the concentration of total cells (10^8 cells/L), X_d is the concentration of death cells (10^8 cells/L), X_v is the concentration of living cells (10^8 cells/L), Glc is the glucose concentration (mM), Glu is the glutamine concentration (mM), Lac is the lactate concentration (mM), Amm is the Ammonium concentration (mM), Ab is the Antibody concentration (mg/L), μ is the specific growth rate (1/d), μ_d is the specific death rate (1/d), q_{glc} is the specific glucose uptake rate (mmol/ 10^8 cells/d), q_{glu} is the specific glutamine uptake rate (mmol/ 10^8 cells/d), k_{sd} is the constant for the spontaneous degradation of glutamine, q_{lac} is the specific lactate production rate (mmol/ 10^8 cells/d), q_{amm} is the specific ammonium production rate (mmol/ 10^8 cells/d) and X_{t0} , X_{d0} , X_{v0} , Glc_0 , Glu_0 , Lac_0 , Amm_0 and Ab_0 are the initial concentrations.

The conversion from glutamine to ammonium can take place in two different ways: the CHO-cells can perform it ($q_{glu} \cdot X_v$, $q_{amm} \cdot X_v$) and it can occur within the medium by spontaneous decomposition ($k_{sd} \cdot Glu$) [38].

No antibodies are produced at the beginning of the experiment hence the differential equation is set to

zero. After a specific change of the environmental conditions, the antibody production starts and Eq. 9 has to be changed to

$$\frac{dAb}{dt} = q_{ab} \cdot X_v \quad \text{Eq. 11}$$

where q_{ab} is the antibody production rate ($\text{mg}/10^8$ cells/d). The supposition is that the decreasing glucose concentration initiates the antibody production. In terms

$$\frac{dAb}{dt} = \begin{cases} 0, & \text{Glc} \geq 14 \text{ mM} \\ q_{ab} \cdot X_v, & \text{Glc} < 14 \text{ mM} \end{cases} \quad \text{Eq. 12}$$

A similar switching situation occurs by the lactate concentration. At the beginning the dynamics are represented by Eq. 7 and after a specific change of the environmental conditions, especially the lactate concentration passes a threshold, the dynamics are described by

$$\frac{dLac}{dt} = \begin{cases} q_{lac} \cdot X_v - q_{lacs} \cdot X_v, & \text{Lac} \geq 19 \text{ mM} \\ q_{lac} \cdot X_v, & \text{Lac} < 19 \text{ mM} \end{cases} \quad \text{Eq. 13}$$

where q_{lacs} is the specific lactate consumption rate ($\text{mmol}/10^8$ cells/d). The glutamine consumption and production, respectively, leads to the following switching equation, whereby the change is initiated by the decreasing glutamine concentration

$$\frac{dGlu}{dt} = \begin{cases} -q_{glu} \cdot X_v - k_{sd} \cdot Glu, & \text{Glu} \geq 0.4 \text{ mM} \\ -q_{glu} \cdot X_v - k_{sd} \cdot Glu + q_{glus} \cdot X_v, & \text{Glu} < 0.4 \text{ mM} \end{cases} \quad \text{Eq. 14}$$

where q_{glus} is the specific glutamine production rate ($\text{mmol}/10^8$ cells/d) and the corresponding dynamics for the ammonium concentration are

$$\frac{dAmm}{dt} = \begin{cases} q_{amm} \cdot X_v + k_{sd} \cdot Glu, & \text{Glu} \geq 0.4 \text{ mM} \\ q_{amm} \cdot X_v + k_{sd} \cdot Glu - q_{amms} \cdot X_v, & \text{Glu} < 0.4 \text{ mM} \end{cases} \quad \text{Eq. 15}$$

where q_{amms} is the specific ammonium consumption rate ($\text{mmol}/10^8$ cells/d).

Figure 14 displays the Petri net modeling the discussed conditions above (Eq. 2-Eq. 5, Eq. 10, Eq. 12-Eq. 15). All places and transitions are continuous. Table 6 contains the places and their corresponding substances and Table 7 summarizes the information of the transitions. The orange Activation-boxes are wrappers of the Reactions sub-library and they work like a discrete switch. When the token number of the connected place exceeds the entered value of the parameter `tres` or fall below the entered value of the parameter `inhi`, the connected transition becomes active and remains active until one of the connected places becomes empty in contrast to the threshold and inhibition values of the transitions.

Everything inside the brown cell mass occurs within the cells and outside of this picture are the reaction for the spontaneous decomposition of glutamine and the substances that the cells release to the medium. The total amount of cells, the sum of living cells and death cells, is modeled by an algebraic equation $X_{t_t} = X_v \cdot t + X_d \cdot t$.

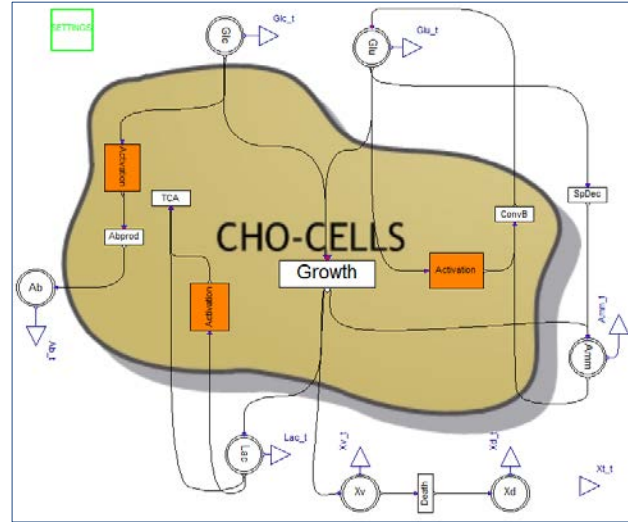


Figure 14: Petri net model of the CHO metabolism in Figure 12

Table 6: Places of the CHO-Model in Figure 14 and the corresponding substances

Place	Substance
Xv	Concentration of living CHO-Cells
Xd	Concentration of death CHO-Cells
Glc	Glucose concentration
Glu	Glutamine concentration
Lac	Lactate concentration
Amm	Ammonium concentration

The experimental data of Figure 13 are approximated by smoothing splines to get further insight to the relations between the respective specific rates. The rates at the beginning of the simulation can be calculated by the following equations

$$\mu = \frac{1}{X_v} \cdot \frac{dX_{total}}{dt} \quad \text{Eq. 16}$$

$$\mu_d = \frac{1}{X_v} \cdot \frac{dX_{death}}{dt} \quad \text{Eq. 17}$$

$$q_{glc} = -\frac{1}{X_v} \cdot \frac{dGlc}{dt} \quad \text{Eq. 18}$$

$$q_{glu} = -\frac{1}{X_v} \cdot \left(\frac{dGlu}{dt} + k_{sd} \cdot Glu \right) \quad \text{Eq. 19}$$

$$q_{lac} = \frac{1}{X_v} \cdot \frac{dLac}{dt} \quad \text{Eq. 20}$$

$$q_{amm} = \frac{1}{X_v} \cdot \left(\frac{dGlu}{dt} - k_{sd} \cdot Glu \right) \quad \text{Eq. 21}$$

The specific antibody production rate can be calculated after day 2.5 when the cells start with the production

$$q_{ab} = \frac{1}{X_v} \cdot \frac{dAb}{dt} \quad \text{Eq. 22}$$

The relations analysis yields the following equation structures for the specific rates

$$\mu = \mu_{max} \cdot \frac{Glu}{K_{Glu} + Glu} \quad \text{Eq. 23}$$

$$\mu_d = \mu_{dmax} \cdot \frac{KD_{Glc}}{KD_{Glc} + Glc} \quad \text{Eq. 24}$$

$$q_{glc} = \frac{1}{Y_{X,Glc}} \cdot \mu \quad \text{Eq. 25}$$

$$q_{glu} = \frac{1}{Y_{X,Glu}} \cdot \mu \quad \text{Eq. 26}$$

$$q_{lac} = Y_{Lac,Glc} \cdot q_{glc} \quad \text{Eq. 27}$$

$$= Y_{Lac,Glc} \cdot \frac{1}{Y_{X,Glc}} \cdot \mu$$

$$q_{amm} = Y_{Amm,Glu} \cdot q_{glu} \quad \text{Eq. 28}$$

$$= Y_{Amm,Glu} \cdot \frac{1}{Y_{X,Glu}} \cdot \mu$$

$$q_{ab} = k_{ab} \quad \text{Eq. 29}$$

$$q_{lacs} = k_{lacs} \quad \text{Eq. 30}$$

$$q_{amms} = k_{amms} \quad \text{Eq. 31}$$

$$q_{glus} = Y_{Glu,Amm} \cdot q_{amms} \quad \text{Eq. 32}$$

with the parameters μ_{max} (1/d) as maximum specific growth rate and K_{Glu} as constant of the Monod kinetics, μ_{dmax} (1/d) as maximum specific death rate, KD_{Glc} as constant of the death kinetics (mM), $Y_{X,Glc}$ (10^8 cells/mmol), $Y_{X,Glu}$ (10^8 cells/mmol), $Y_{Lac,Glc}$ (mol/mol), $Y_{Amm,Glu}$ (mol/mol) and $Y_{Glu,Amm}$ (mol/mol) as yield coefficients, k_{ab} (mg/ 10^8 cells) as constant of the antibody production, k_{lacs} (mmol/ 10^8 cells) as constant of the lactate consumption and k_{amms} as constant of the ammonium consumption.

For performing a sensitivity analysis and afterwards a parameter optimization for the 13 model parameters, the Petri net model in Dymola (Figure 14) has to be connected to Matlab via a Simulink interface as described in section 4.1. The simulation results of all token numbers are needed in Matlab, thus all places have a blue RealOutput connector (Figure 14) so that a corresponding port at the Simulink interface is provided.

Table 7: Transitions of the CHO-Model in Figure 14 and the corresponding reactions and the edge weighting functions (Eq. 2-Eq. 5, Eq. 10, Eq. 12-Eq. 15)

Transition	Reaction	Weightings		Conditions
Growth	Cell growth	Glc → Growth	$q_{glc} \cdot X_v \cdot t$	
		Glu → Growth	$q_{glu} \cdot X_v \cdot t$	
		Growth → Xv	$\mu \cdot X_v \cdot t$	
		Growth → Lac	$q_{lac} \cdot X_v \cdot t$	
		Growth → Amm	$q_{amm} \cdot X_v \cdot t$	
Death	Cell death	Xv → Death	$\mu_d \cdot X_v \cdot t$	
		Death → Xd	$\mu_d \cdot X_v \cdot t$	
TCA	Lactate consumption	Lac → TCA	$q_{lacs} \cdot X_v \cdot t$	Activation Box thres=19
Abprod	Antibody production	Abprod → Ab	$q_{ab} \cdot X_v \cdot t$	Activation Box inhi=14
SpDec	Spontaneous decomposition of glutamine to ammonium	Glu → SpDec	$k_{sd} \cdot Glu \cdot t$	
		SpDec → Amm	$k_{sd} \cdot Glu \cdot t$	
ConvB	Conversion of ammonium back to glutamine	Amm → ConvB	$q_{amms} \cdot X_v \cdot t$	Activation Box inhi=0.4
		ConvB → Glu	$q_{glus} \cdot X_v \cdot t$	

Before the 13 parameters of the model are estimated a global sensitivity analysis is performed to get further insight in the parameter characteristics. This analysis is the basis of the following parameter optimization since less sensitive parameters can be fixed during the optimization process to increase the chance of a converging optimization algorithm. The global sensitivity analysis is performed by Matlab with eFAST method explained in section 4.2. Therefore, the model is simulated several times with different parameter settings and each time the objective function in Eq. 1 is evaluated. The eFAST-method measures the contribution of each parameter to the variance of this objective function, whereby the parameters are varied in a specific range. If a parameter contributes less to the variance, this parameter cannot be identified within an optimization procedure and has to be fixed on the other hand if a parameter contributes much to the variance of the objective function this parameter is identifiable.

The results of the global sensitivity analysis, i.e. the contribution of each parameter to the variance of the objective function, are displayed in Figure 15. It becomes clear that 7 of 13 parameters contribute 91 % of the variance so that 6 parameters

$Y_{X,Glu}, \mu_{dmax}, KD_{glc}, k_{lacs}, Y_{Lac,Glc}, Y_{Amm,Glu}$ can be fixed during the optimization process and 7 $k_{sd}, Y_{Glu,Amm}, k_{amms}, Y_{X,Glc}, \mu_{max}, K_{Glu}, k_{ab}$ have to be optimized.

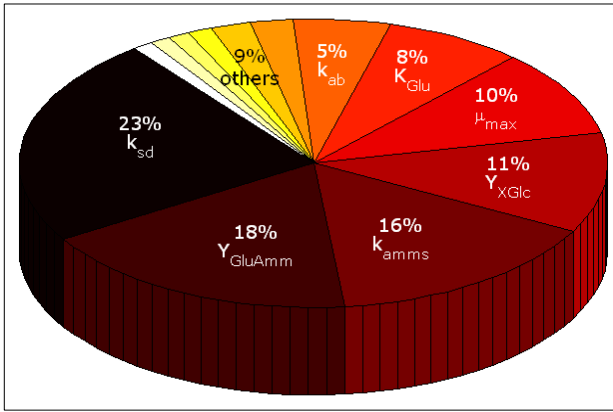


Figure 15: Model variance contribution of every parameter according to the eFAST method

The parameter optimization is performed by CMA-ES method explained in section 4.3. The optimization procedure takes place in Matlab via a Simulink interface. Figure 16 displays the results of this optimization procedure which show a good agreement with the experimental data.

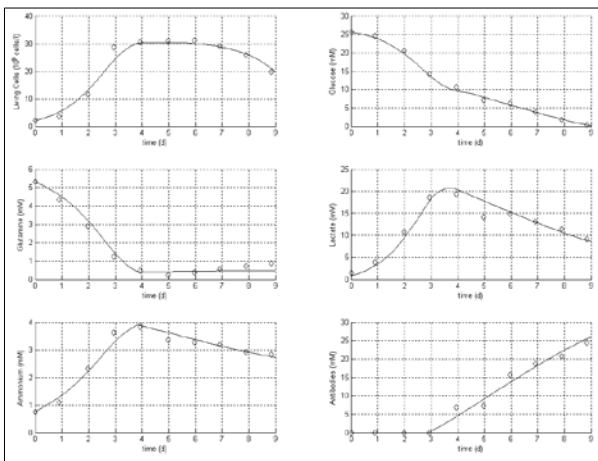


Figure 16: Results of the parameter optimization procedure

To achieve a good model of the CHO-metabolism, it is also possible to choose a stochastic approach, i.e. a stochastic Petri net model and a stochastic simulation according to Gillespie's algorithm as described in section 4.4. The edge weightings of the continuous approach in Table 7 are now the dynamic values of the characteristic parameter λ of the exponential distribution by which the delay of the stochastic transition is chosen randomly at every activation point in time (cp. Section 2). The transformation of the parameters of the continuous to the stochastic model is well studied and can be found in [39]. Figure 17 displays the CHO-metabolism modeled by a stochastic Petri net, whereby the places are discrete and the transitions are stochastic. The tokens represent here different concentration levels like it is presented in [14]. One token equals to 0.5 (mM,

10^8Cells/l , mg/l), thus there are $N + 1 = 90 + 1$ different levels since the maximum concentration (M) is set to 45. The values of M and N can be entered in the green settings-box which has to be a part of every model and can be found in the Global-library. This stochastic Petri net model is also connected to a Simulink interface in Matlab so that the stochastic simulation can take place within an m-file. The results are displayed in Figure 18 where 500 Simulation are accomplished and the means were built with 10 simulations, respectively.

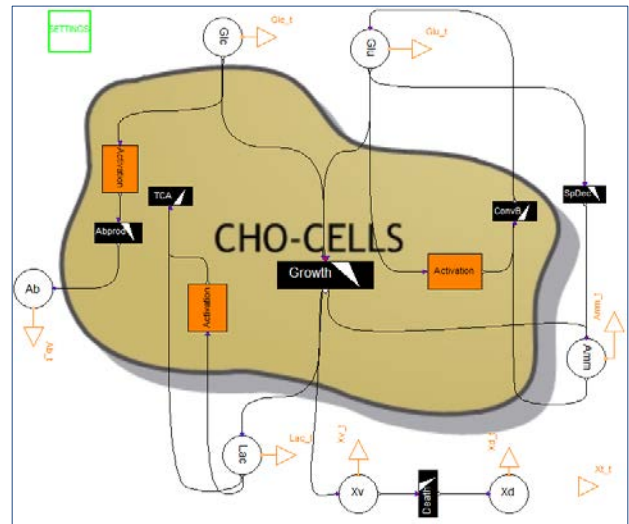


Figure 17: Stochastic Petri Net of the main CHO-metabolism in Figure 12

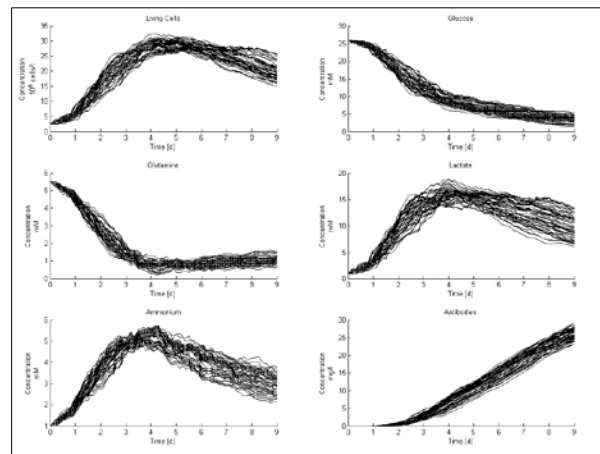


Figure 18: The stochastic simulation results according to Gillespie's algorithm of the stochastic Petri Net model in Figure 17

6 Conclusions

The Petri net library in Modelica is a good instrument for hybrid modeling of biological systems. The advantages of this approach are:

- The object-oriented modeling language Modelica is able to model discrete places and transitions as well as stochastic and continuous ones. The places and transitions are models that easily can be changed, modified, or expanded so that further Petri net extensions can be implemented fast.
- The language allows the realization of hybrid models by combining discrete and continuous processes. The hybrid simulation with discrete events and the solution of continuous differential equations is then performed by Dymola or by another Modelica-tool.
- The Reactions sub-library offers a fast and simple way to build up a model and further reactions can be easily added.
- The hierarchical modeling concept of Modelica enables a structuring of the models on different levels which is useful when the model is complex and used by different persons with different aims.
- The Petri net animation of Dymola offers a way to get insight of the token distribution of large and complex Petri nets.
- The coupling of Dymola-models and Simulink-models allows the simulation of a model many times and use the arising simulation results for subsequent calculations so that stochastic simulation, sensitivity analysis and parameter identification in Matlab is possible.
- The Petri net library can be integrated in other Petri net modeling tools by parsing the Petri net of the respective tool (e.g. XML-format) to Modelica-text and simulate it via a batch process where the simulation results are saved in a data file.

In this manner the new Petri net library in combination with Matlab Simulink leads to a complete environment for hybrid modeling of biological systems.

References

- [1] V.N. Reddy, M.L. Mavrovouniotis, M.N. Liebman (Eds.), Petri net representations in metabolic pathways: Proc Int Conf Intell Syst Mol Biol, 1993.
- [2] C.A. Petri, Communication with automata, Rome Air Development Center, Research and Technology Division, 1966.
- [3] R. Valk, Self-modifying nets, a natural extension of Petri nets, Automata, Languages and Programming (1978) 464–476.
- [4] R. Hofestädt, S. Thelen, Quantitative modeling of biochemical networks, In Silico Biology 1 (1998) 39–53.
- [5] F. Bause, P.S. Kritzinger, Stochastic Petri Nets, Vieweg, 2002.
- [6] M. Heiner, D. Gilbert, R. Donaldson, Petri nets for systems and synthetic biology, Formal Methods for Computational Systems Biology (2008) 215–264.
- [7] D. Gilbert, M. Heiner, From Petri nets to differential equations-an integrative approach for biochemical network analysis, Petri Nets and Other Models of Concurrency-ICATPN 2006 (2006) 181–200.
- [8] A. Doi, S. Fujita, H. Matsuno, M. Nagasaki, S. Miyano, Constructing biological pathway models with hybrid functional Petri nets, In Silico Biology 4 (2004) 271–291.
- [9] P.J. Mosterman, M. Otter, H. Elmqvist, Modeling Petri nets as local constraint equations for hybrid systems using Modelica, Citeseer, Reno, Nevada, Proceedings of SCS Summer Simulation Conference, 1998, pp. 314–319.
- [10] S.M. Fabricius, Extensions to the Petri Net Library in Modelica, ETH Zurich, Switzerland (2001).
- [11] M. Otter, K.E. Årzén, I. Dressler (Eds.), StateGraph-a Modelica library for hierarchical state machines: 4th International Modelica Conference, 2005.
- [12] S. Proß, B. Bachmann, A Petri Net Library for Modeling Hybrid Systems in OpenModelica, Como, Italy, Modelica Conference proceedings, 2009.
- [13] Modelica Association, Modelica - A Unified Object-Oriented Language for Physical Systems Modeling Language Specification Version 3.2 (2010).
- [14] D. Gilbert, M. Heiner, S. Lehrack (Eds.), A unifying framework for modelling and analysing biochemical pathways using Petri nets: Proceedings of the 2007 international conference on Computational methods in systems biology, Springer-Verlag, 2007.
- [15] S. Proß, B. Bachmann, R. Hofestädt, K. Niehaus, R. Ueckerdt, F.J. Vorhölter, P. Lutter, Modeling a Bacterium's Life: A Petri-Net Library in Modelica, Como, Italy, Modelica Conference proceedings, 2009.
- [16] Dynasim AB, Dymola-Dynamic Modeling Laboratory-User Manual Volume 2, Lund/Sweden, 2010.
- [17] A. Saltelli, Sensitivity analysis in practice: a guide to assessing scientific models, John Wiley & Sons Inc, 2004.
- [18] A. Saltelli, M. Ratto, T. Andres, Global sensitivity analysis: the primer, John Wiley & Sons Ltd, 2008.

- [19] A. Saltelli, K. Chan, E.M. Scott, Sensitivity analysis, Wiley New York, 2000.
- [20] R.I. Cukier, C.M. Fortuin, K.E. Shuler, A.G. Petschek, J.H. Schaibly, Study of the sensitivity of coupled reaction systems to uncertainties in rate coefficients. I Theory, *The Journal of Chemical Physics* 59 (1973) 3873–3876.
- [21] J.H. Schaibly, K.E. Shuler, Study of the sensitivity of coupled reaction systems to uncertainties in rate coefficients. II Applications, *The Journal of Chemical Physics* 59 (1973) 3879–3888.
- [22] R.I. Cukier, J.H. Schaibly, K.E. Shuler, Study of the sensitivity of coupled reaction systems to uncertainties in rate coefficients. III. Analysis of the approximations, *The Journal of Chemical Physics* 63 (1975) 1140–1149.
- [23] A. Saltelli, S. Tarantola, K.P. Chan, A quantitative model-independent method for global sensitivity analysis of model output, *Technometrics* 41 (1999) 39–56.
- [24] R.I. Cukier, H.B. Levine, K.E. Shuler, Nonlinear sensitivity analysis of multiparameter model systems, *Journal of Computational Physics* 26 (1978) 1–42.
- [25] M. Koda, G.J. Mcrae, J.H. Seinfeld, Automatic sensitivity analysis of kinetic mechanisms, *Int. J. Chem. Kinet.* 11 (1979) 427–444.
- [26] J. Nocedal, S.J. Wright, Numerical optimization, Springer-Verlag New York Inc, New York, Berlin, Heidelberg, 1999.
- [27] T. Weise, *Global Optimization Algorithms – Theory and Application*, 2009. <http://www.it-weise.de/>.
- [28] T. Bäck, *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*, Oxford University Press, USA, 1996.
- [29] N. Hansen, *The CMA evolution strategy: a comparing review*, *Towards a new evolutionary computation* (2006) 75–102.
- [30] D.T. Gillespie, Exact stochastic simulation of coupled chemical reactions, *The journal of physical chemistry* 81 (1977) 2340–2361.
- [31] Birch, JR, A.J. Racher, Antibody production, *Advanced drug delivery reviews* 58 (2006) 671–685.
- [32] N. Kurano, C. Leist, F. Messi, S. Kurano, A. Fiechter, Growth behavior of Chinese hamster ovary cells in a compact loop bioreactor. 2. Effects of medium components and waste products, *Journal of biotechnology* 15 (1990) 113–128.
- [33] M.S. Lao, D. Toth, Effects of ammonium and lactate on growth and metabolism of a recombinant Chinese hamster ovary cell culture, *Biotechnology progress* 13 (1997) 688–691.
- [34] S.S. Ozturk, M.R. Riley, B.O. Palsson, Effects of ammonia and lactate on hybridoma growth, metabolism, and antibody production, *Biotechnol. Bioeng.* 39 (1992) 418–431.
- [35] Y.S. Tsao, A.G. Cardoso, R.G. Condon, M. Voloch, P. Lio, J.C. Lagos, B.G. Kearns, Z. Liu, Monitoring Chinese hamster ovary cell culture by the analysis of glucose and lactate metabolism, *Journal of biotechnology* 118 (2005) 316–327.
- [36] A. Provost, G. Bastin, S.N. Agathos, Y.J. Schneider, Metabolic design of macroscopic bioreaction models: application to Chinese hamster ovary cells, *Bioprocess and biosystems engineering* 29 (2006) 349–366.
- [37] J. Link, *Charakterisierung der Prozessparameter tierischer Zellkulturen in Schüttelinkubatoren*. Bachelor thesis, Bielefeld, 2010.
- [38] S.S. Ozturk, B.O. Palsson, Chemical decomposition of glutamine in cell culture media: effect of media type, pH, and serum concentration, *Biotechnology progress* 6 (1990) 121–128.
- [39] D.J. Wilkinson, *Stochastic modelling for systems biology*, Chapman & Hall/CRC, 2006.

Modelling of a Chemical Reactor for Simulation of a Methanisation Plant

Bader, A.¹ Bauersfeld, S.¹ Brunhuber, C.² Pardemann, R.¹ Meyer, B.¹

¹ Technische Universität Bergakademie Freiberg
Department of Energy Process Engineering and Chemical Engineering
Fuchsmühlenweg 1, Haus 1, 09599 Freiberg, Germany

² Siemens AG, Energy Solutions
Freyeslebenstraße 1, 91058 Erlangen, Germany

Abstract

The chemical and physical modelling and transient simulation of a plant with chemical reactors can be useful within dimensioning, optimisation, operation studies and analysing of time critical processes. Therefore, a reactor model for thermodynamic equilibrium conditions has been implemented. The Model is based on the free *Modelica Fluid* library and contains correlations for heat and mass transfer and pressure drop. The model contains the components: H₂, CO, CO₂, H₂O, CH₄, N₂

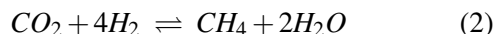
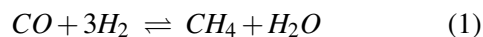
Keywords: chemical reactor; thermodynamic equilibrium; part load; CO₂; CO; H₂; CH₄; H₂O

1 Introduction

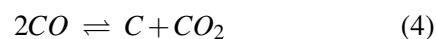
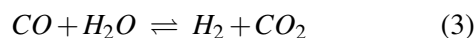
It is well known that the oil and natural gas reserves are limited. The production of fuels is based almost completely on oil and natural gas. Hence there is the wish to extend availability of a secure energy supply. One solution is the conversion of coal or biomass to synthetic or substitute natural gas (SNG) by gasification and methanation. The reserves of coal will persist more than 150 years and biomass is a renewable energy source. The modelling of the SNG synthesis focus only on the methanation step. The simulation of a methanisation plant is in interest for plant manufacturers and operators to optimise the construction and the operation. Further interest are studies of critical processes for understanding of complex physical and chemical processes in order to increase the reliability and availability of the plant during part load and plant trips.

2 The SNG synthesis

The raw gas for the SNG synthesis contains mainly H₂, CO, CO₂, H₂O, CH₄, N₂. This composition of the raw gas depends on the gasification technology. The SNG synthesis is a heterogeneously catalysed process. During the methanation, the following chemical reactions execute. The hydrogenation of carbon oxides to methane are the so called CO methanation reaction in equation 1 and the CO₂ methanation reaction in equation 2.



Two further independent reactions are important: the CO-Shift reaction in equation 3 and the Boudouard reaction in equation 4.



All the reactions of the methanation are exothermic, see Table 1. The methanation is favoured at low temperatures. Furthermore the methanation is benefited at high pressure, as the reaction 1 and 2 execute with volume decreasing.

The major criteria of catalysed methanation reactors is to achieve efficient removal of heat. The first reason is to minimise catalyst deactivation due to thermal stress. The second reason is to avoid a limitation in the methane yield due to approaching the chemical equilibrium. The Topsøe recycle energy-efficient methanation process (TREMP) from Haldor Topsøe is

Table 1: Reaction heat at standard conditions

reaction	reaction heat, $\Delta_R H^0$ kJ/mol
1	-206
2	-165
3	-41
4	-172

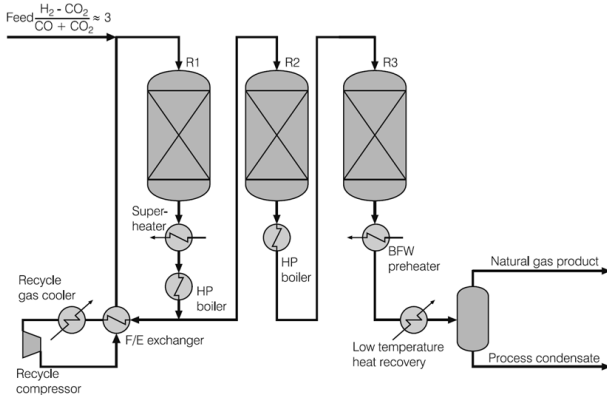


Figure 1: Flowsheet of the Topsøe recycle energy-efficient methanation process [1]

a suitable reactor concept for the production of SNG [1], which is used for the modelling.

The TREMP consists of 3 adiabatic fixed bed reactors with gas recycle cooling and interstage cooling. Efficient recovery of the reaction heat is essential for the industrial methanation technology. That's why a counter current water cycle stream is used for the interstage cooling. The water cycle is simultaneously used for high pressure steam generation. The gas recycle at the first reactor reduces the yield and the temperature in the reactor.

3 Developed models

The aim of the modelling is to implement a library with physical based models of components of the TREMP methanation plant. The library enables investigations with models of a reactor, a heat exchanger, a simple pump, a flash, a gas and water splitter. The implemented library is based on the free Modelica Fluid library, which offers a base with respect to the implementation of the three balance equations and the interaction through the fluid ports.

3.1 Reactor

It is necessary to calculate the thermodynamic equilibrium for the methanation reactor yield, as it will be

 Table 2: Parameters A and B of equation 10

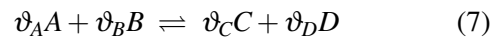
reaction	A	B
1	-29.3014	26248.4
3	-4.3537	4593.2

achieved both in the full load and the part load of the methanation plant, Harms in [2]. The thermodynamic equilibrium constant K_a is used in relation with the law of mass action to determine the molar fractions of the components and the yield of the reactions at the thermodynamic equilibrium. K_P is the equilibrium constant of the partial pressures, p_i , of the species, which is connected to the K_a with the fugacity coefficients, K_α , in equation 5.

$$K_P = \frac{K_a}{K_\alpha} \quad (5)$$

The law of mass action for the K_P is described in equation 6 for a chemical reaction like in equation 7.

$$K_P = \frac{p_C^{\vartheta_C} \cdot p_D^{\vartheta_D}}{p_A^{\vartheta_A} \cdot p_B^{\vartheta_B}} \quad (6)$$



The equilibrium constant of the molar fractions (K_X) is necessary to calculate the chemical equilibrium. K_X can be computed with K_P , the total pressure, p , and the sum of stoichiometry coefficients of the chemical reaction, how it is shown in equation 8. The molar fraction of the species, X_i , can be determined with equation 9.

$$K_X = K_P \cdot p^{-\sum \vartheta_i} \quad (8)$$

$$K_X = \frac{X_C^{\vartheta_C} \cdot X_D^{\vartheta_D}}{X_A^{\vartheta_A} \cdot X_B^{\vartheta_B}} \quad (9)$$

It has to be noted that the thermodynamic equilibrium constant is temperature dependent. The temperature can be approximated with equation 10.

$$\ln(K_a) = A + \frac{B}{T} \quad (10)$$

The parameters A and B of equation 10 are given in Table 2.

Two simplifications were assumed for the calculation of the thermodynamic equilibrium. The first is the assumption of ideal gas law for the species. Therewith the fugacity coefficients can be neglected in equation 5, $K_\alpha = 1$.

The second simplification is the neglect of the Boudouard reaction, equation 4, which occurs if the stoichiometry ratio of equation 11 is lower than 3. In that case, carbon is produced as a product of reaction 4. Carbon leads to catalyst deactivation by forming of carbon deposition on the catalyst surface. This is the technical aspect to eliminate reaction 4.

$$\frac{H_2 - CO_2}{CO + CO_2} \geq 3 \quad (11)$$

If the stoichiometry ratio from the reactants of equation 11 is at least 3 or more, CO reacts with H₂ completely to CH₄ and H₂O according to equation 1. The stoichiometry ratio of equation 11 can be adjusted by converting CO with H₂O to CO₂ and H₂ as long as the H₂O is high enough, see Anderlohr in [3]. Finally the system of methanation reactions can be reduced to reaction 1 and 3, as the stoichiometry ratio of equation 11 for all the gas streams in the TREMP are higher than three. Therewith the molar fraction of the species can be determined with the equations from 12 to 17, in which 0 stands for the start state and 1 for the equilibrium state. The yield U_1 and U_2 is the yield of the reactions 1 and 3, whose sum is the total yield of CO, see equation 18.

$$X_{H_2,1} = \frac{C_{H_2,0} - 3C_{CO,0} \cdot U_1 + C_{CO,0} \cdot U_2}{1 - 2 \cdot C_{CO,0} \cdot U_1} \quad (12)$$

$$X_{CH_4,1} = \frac{C_{CH_4,0} + C_{CO,0} \cdot U_1}{1 - 2 \cdot C_{CO,0} \cdot U_1} \quad (13)$$

$$X_{H_2O,1} = \frac{C_{H_2O,0} + C_{CO,0} \cdot U_1 - C_{CO,0} \cdot U_2}{1 - 2 \cdot C_{CO,0} \cdot U_1} \quad (14)$$

$$X_{CO,1} = \frac{C_{CO,0} - C_{CO,0} \cdot U_1 - C_{CO,0} \cdot U_2}{1 - 2 \cdot C_{CO,0} \cdot U_1} \quad (15)$$

$$X_{CO_2,1} = \frac{C_{CO_2,0} + C_{CO,0} \cdot U_2}{1 - 2 \cdot C_{CO,0} \cdot U_1} \quad (16)$$

$$X_{N_2,1} = \frac{C_{N_2,0}}{1 - 2 \cdot C_{CO,0} \cdot U_1} \quad (17)$$

$$U_1 + U_2 = \frac{C_{CO,0} - C_{CO,1}}{C_{CO,0}} \quad (18)$$

In order to determine the chemical equilibrium at known temperature and pressure, a non-linear equation system of the above mentioned equations need to be solved. The results of an example calculation are given in Figure 2.

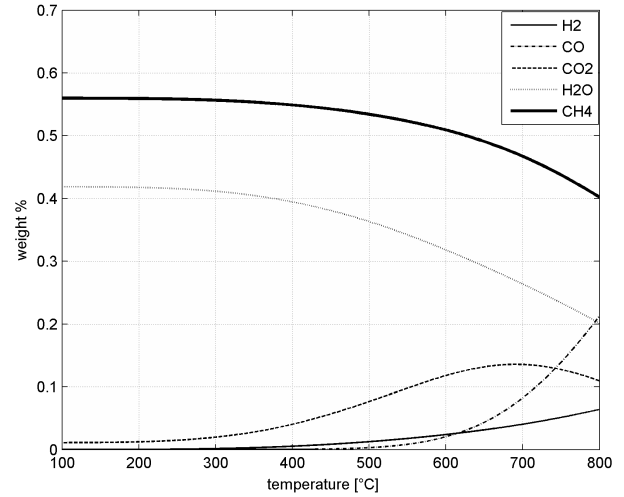


Figure 2: Thermodynamic equilibrium for the feed gas in the range of 100-800 °C at 30 bar, start composition in w.%: 4,97 H₂, 13,80 CO, 11,97 CO₂, 24,11 H₂O, 44,12 CH₄, 1,03 N₂;

Table 3: Parameters A, B and C of equation 20

reaction	A	B	C
1	0.0266	-47.7331	-205094.5788
3	0.0026	-7.4437	-41557.3842

The energy balance need to be completed by the produced reaction heat which influence the equilibrium temperature. The amount of the produced reaction heat is the higher the yield of the reactions is. A high reaction heat leads to high temperatures. But the higher the temperature, the lower is the yield. The reaction heat, \dot{Q}_{Rkt} , can be determined by equation 19, in which Δh is the specific enthalpy of the reaction and M_{gas} the mean molar mass of the gas. The temperature dependency of the specific reaction enthalpy is approximated with equation 20. The parameters of equation 20 for reaction 1 and 3 are given in Table 3.

$$\dot{Q}_{Rkt} = (\Delta h_{R-1} \cdot U_1 + \Delta h_{R-3} \cdot U_2) \cdot \dot{m} \cdot \left(\frac{X_{CO,1}}{M_{gas,1}} - \frac{X_{CO,0}}{M_{gas,0}} \right) \quad (19)$$

$$\Delta h = A \cdot T^2 + B \cdot T + C \quad (20)$$

The configuration of the reactor model is shown in Figure 3. The model components are sNG_reaction model, volume, heat capacity, temperature sensor and linear valve with a constant block. The calculation of the thermodynamic equilibrium occurs in the sNG_reaction model. The volume represents the volume of the reactor. The heat capacity model is the heat

capacity of the catalyst bed and the reactor wall. The valve model is used to simulate the pressure drop of gas flow through the catalyst bed.

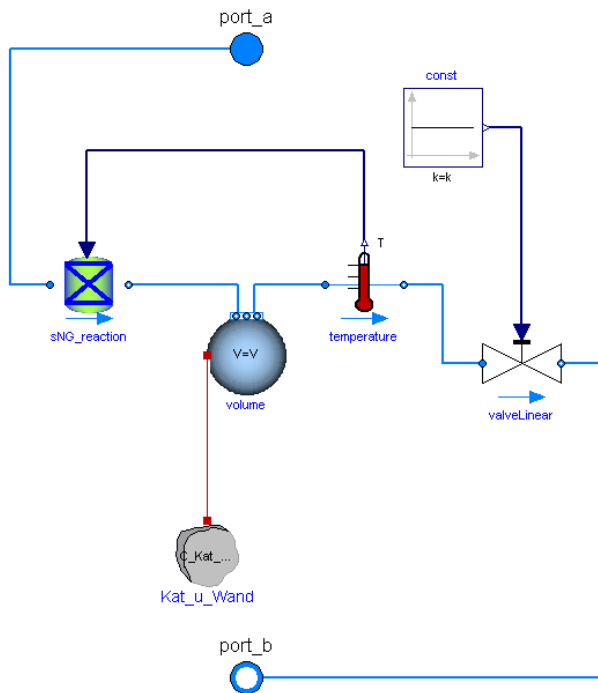


Figure 3: Configuration of the reactor model

The reactor model has three main parameters: volume, pressure drop and heat capacity. The parameters have different impact of the reactor performance. The greater the reactor volume, the later will be the steady state achieved. The pressure drop changes the pressure and therewith the reactor yield. The pressure drop is predefined by the user. The impact of the heat capacity is more difficult. The reactor model need be splitted in series identical reactor zones, which are duplicates of the reactor model configuration in Figure 3, at high heat capacities. The higher the heat capacity, the more zones are needed to get the steady state for temperature and methane at the same time, which is physically necessary, because the temperature influence the chemical equilibrium. The temperature would be change slower than the chemical equilibrium, if the number of zones is to low. An example of the transient temperature behaviour for an 5 zone reactor model at heat capacities in the range of 0 to 10 MJ/K is given in Figure 4. The consequently methane concentration is illustrated in Figure 5.

The reactor ignition or extinction can not be simulated, because no attention is paid to kinetic effects of the chemical reactions.

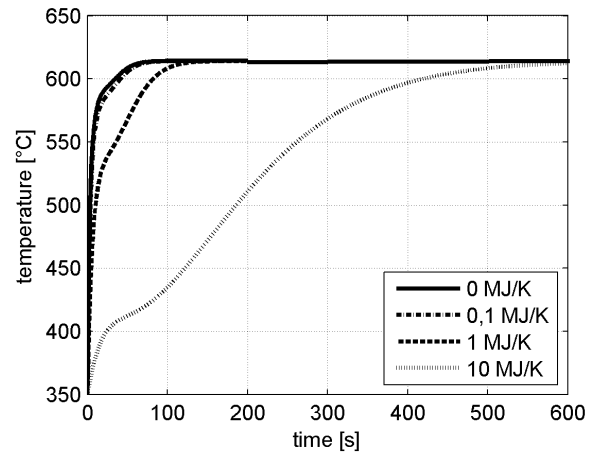


Figure 4: Temperature for a reactor model with 5 zones at heat capacities in the range of 0 to 10 MJ/K

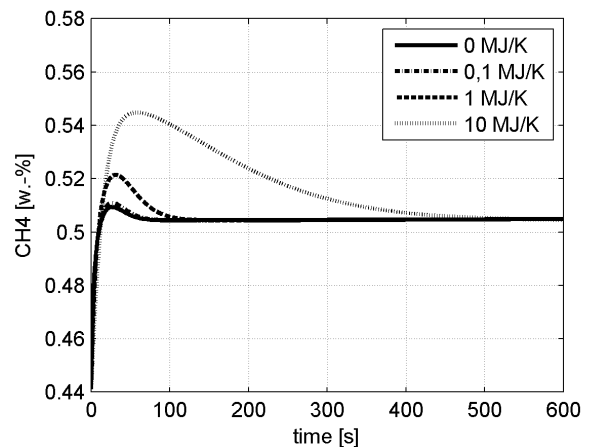


Figure 5: Methane concentration for a reactor model with 5 zones at heat capacities in the range of 0 to 10 MJ/K

3.2 Heat exchanger

The second important model for the modelling of the methanisation plant is the heat exchanger, which is assumed with a simple design. The heat exchanger design comprises of an outside tube, filled with hot SNG gas, which has an inside tube filled with water, how it is shown in Figure 6. The configuration of a heat exchanger zone is given in Figure 7. The zone consists of two volume models. One volume model contains SNG gas and the other water/steam. The volume model are connected by a thermal conduction model. The heat exchanger comprises of such series zones. The water and the gas are in counter current flow as the fluid ports are setted up in this way. Multiple zones are necessary therewith the outflow of the cold side can be hotter than the outflow of hot side, in certain cases. Further-

more, a temperature profil can be generated across the heat exchanger because of the multiple zones.

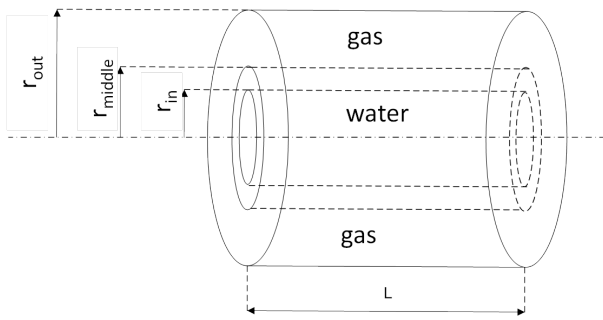


Figure 6: Heat exchanger design

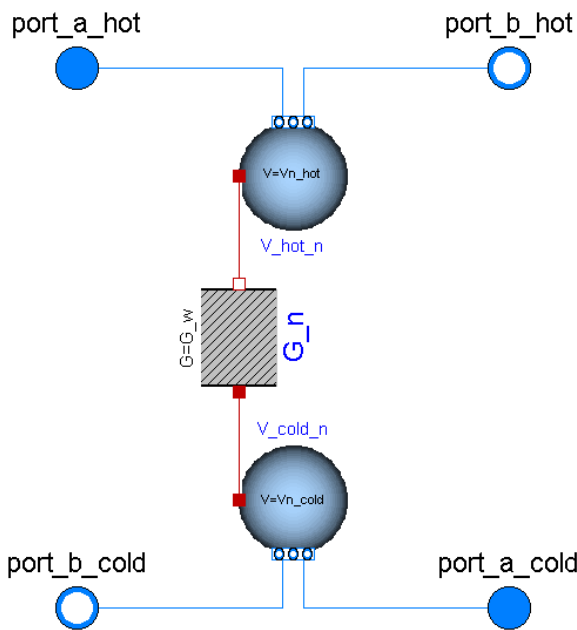


Figure 7: Configuration of a heat exchanger zone

The steady state results of an example simulation are given in Figure 8. The water temperature is constant in the beginning part of the heat exchanger, as the boiling point is achieved at 318 °C at 110 bar. The gas temperature is cooled down continuously.

4 Validation

The steady state results from the simulation will be validated in the following section as well from the single reactor model as the complex model of the methanation plant. The results are compared to two references. The first one are the experimental results of the TREMP which are published by Harms in [2]. The second reference are steady state simulation results with the software ChemCAD which assumes the

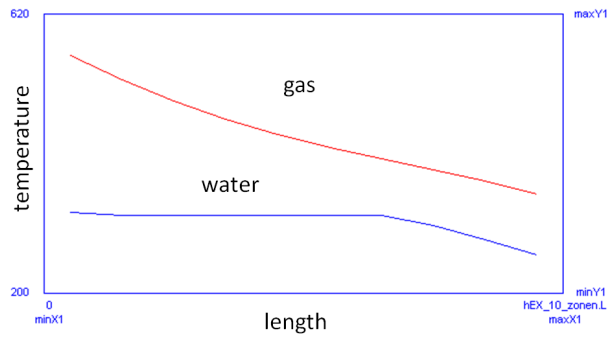


Figure 8: Temperatures of the heat exchanger model at steady state

chemical equilibrium. The results of the reactor model are compared with the data of these two references in Table 4 and of the TREMP model in Table 5. The mistake of the species is always lower than 4 % except of hydrogen at low concentration, where the mistake is under certain conditions up to 10 %.

Table 4: Comparison of the outlet gas composition of the first reactor of the TREMP with the reactor model in Modelica and in ChemCAD. Inlet gas: composition in weight %: 4,97 H₂, 13,80 CO, 11,97 CO₂, 24,11 H₂O, 44,12 CH₄, 1,03 N₂; temperature 300 °C; pressure 30 bar

	Unit	Outlet TREMP	Outlet Modelica	Outlet ChemCAD
Temp.	°C	600	600	599
H ₂	w.%	2,47	2,40	2,41
CO	w.%	2,01	2,04	1,97
CO ₂	w.%	12,13	11,79	11,92
H ₂ O	w.%	31,48	31,82	31,76
CH ₄	w.%	50,82	50,92	50,91
N ₂	w.%	1,10	1,03	1,03

5 Transient behaviour

The reactor and the heat exchanger model, see section 3, are used to build up the model of the methanation plant like it is shown in Figure 1. The model of the methanation plant is utilised for transient simulation studies. In the following, results of a part load simulation are presented. The simulation of the load change is faster than realtime. In order to realise the part load, the feed mass stream was reduced to 50 % after the TREMP model is steady state which is after 300 s simulation time, see Figure 9. The steady state in part load is achieved after the simulation time 1000 s.

Table 5: Comparison of the outlet gas composition of the first reactor of the TREMP with the TREMP model in Modelica and in ChemCAD. Inlet gas: composition in weight %: 13,08 H₂, 51,76 CO, 11,49 CO₂, 0,00 H₂O, 22,54 CH₄, 1,13 N₂; temperature 150 °C; pressure 30 bar. Recycle rate = 3,3

	Unit	Outlet TREMP	Outlet Modelica	Outlet ChemCAD
Temp.	°C	600	596	594
H ₂	w.%	2,47	2,34	2,34
CO	w.%	2,01	1,91	1,83
CO ₂	w.%	12,13	11,64	11,73
H ₂ O	w.%	31,47	31,94	31,91
CH ₄	w.%	50,82	51,04	51,06
N ₂	w.%	1,10	1,13	1,13

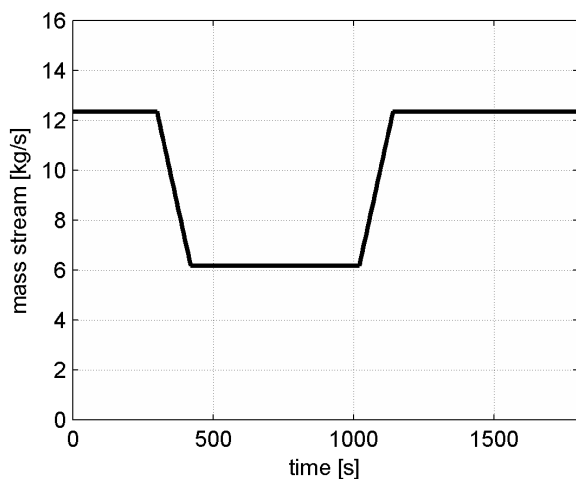


Figure 9: Mass stream of the feed in the Modelica TREMP model

The result are lower gas temperatures in the reactors which leads to higher methane concentrations in the gas, how it is shown in Figure 10 and 11. The feed mass stream rise back up to 100 % after 1000 s simulation time to full load. The steady state is got after simulation time 2000 s. The temperature and the methane concentration achieve the same level as before the part load.

Finally the gas product quality at part load is almost equal to the full load. Furthermore, it is still possible to produce steam, but the pressure of the steam need to be adjusted.

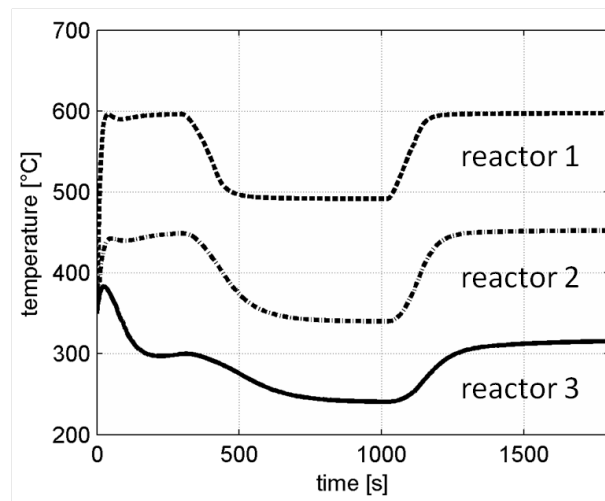


Figure 10: Temperatures in the reactors in the Modelica TREMP model

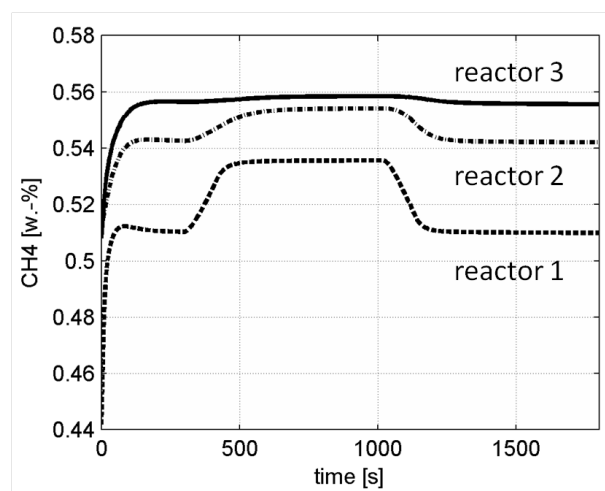


Figure 11: Methane concentration in the reactors in the Modelica TREMP model

6 Conclusion

A succesfull generation of a dynamic methanation plant model could be realised. Therefore the development of a simplified reactor and heat exchanger model was necessary. The results of the reactor plant and the plant model of the TREMP could be validated successful of steady state performance with ChemCAD simulations and experimental data from Harms [2]. The dynamic behaviour could be only validated by a plausibility check.

The TREMP plant model is finally useful for the development and check of control concepts, and furthermore for the analysis of trip scenarios and load changes.

7 Outlook

It has to be mentioned that the solution of the nonlinear equation system takes a long calculation time of the CPU, because the slow mixed implicit/explicit euler method is used for simulation. All the other integrated numerical solution algorithms in Dymola lead to instability. Hence, a faster and more stable solution algorithm for the calculation of the thermodynamic equilibrium is necessary. One proposal is to work with look up tables which are precalculated for certain reactor yields. The disadvantage of this solution is a reduced flexibility of the model. Another proposal is the use of the so called method: minimisation of the gibbs enthalpie, for the calculation of the thermodynamic equilibrium, which is published by White et al. in [4]. At this method solves mainly derivatives, the equation system may be solved much faster and more numeric robust.

Moreover, a kinetic model is necessary to estimate the limits for the parameter therewith the assumption of the thermodynamic equilibrium can be justified.

References

- [1] Haldor Topsoe: From solid fuels to substitute natural gas (SNG) using TREMP, <http://www.topsoe.com>
- [2] Harms, H.; Höhle, B.; Skov, A.; Methanisierung kohlenmonoxidreicher Gase beim Energietransport. In: Chemie.-Ing. Technik, (6) page 504-515, 1980.
- [3] Anderlohr, A.; Untersuchung zu gleichzeitigen Methanisierung und Konvertierung von CO-reicher Gase in einer katalytischen Wirbelschicht. Karlsruhe, Germany: PhD thesis, Fakultät für Chemieingenieurwesen, Tech. Hochschule Karlsruhe, 1979.
- [4] White, W.; Johnson, S., Danzig, G.: Chemical Equilibrium in Complex Mixtures. In: The Journal of Chemical Physics 28(5), S. 751-755 (1958)

Modelling of System Properties in a Modelica Framework

Audrey Jardin Daniel Bouskela Thuy Nguyen Nancy Ruel
EDF R&D, STEP Department
6 quai Watier, 78401 CHATOU Cedex, FRANCE
audrey.jardin@edf.fr daniel.bouskela@edf.fr nancy.ruel@edf.fr

Eric Thomas Laurent Chastanet
Dassault-Aviation
78 quai Marcel Dassault Cedex 300, 92552 St CLOUD Cedex, FRANCE
eric.thomas@dassault-aviation.com laurent.chastanet@dassault-aviation.com

Raphaël Schoenig Sandrine Loembé
Dassault-Systèmes
10 rue Marcel Dassault, 78946 VELIZY-VILLACOUBLAY Cedex, FRANCE
raphael.schoenig@3ds.com sandrine.loembe@3ds.com

Abstract

In order to improve the engineering processes and especially the corresponding verification and validation phases, this article deals with the modeling of system properties in a Modelica framework. The term “property” is intended here to be generic and refers to a system requirement or limitation as well as a validity domain of a model. The choice of the Modelica language is justified by a desire to use its equation-based feature to model system properties in an unambiguous and explicit way. Besides, choosing only one formalism to describe the system properties and the physical equations of the model should ease the expression of the model validity domains.

After having introduced several theoretical concepts to formally describe a system property, the development of a dedicated library is explained and illustrated on an industrial example taken from the aeronautics domain. Some checks of system properties are thus performed by co-simulating behavioral and properties models. Finally, some extensions of the Modelica language are advocated in order to improve the applicability range and efficiency of properties modeling for complex systems, and especially to increase the rigor of their validations by enabling formal proofs.

Keywords: Modelling; Checking; Property; Modelica.

1 Introduction

The study of performance and safety is today of prime interest when designing complex systems. At each stage of the design cycle, engineers should check the conformance of their technical choices with respect to the initial specifications. In such a Verification & Validation (V&V) process, the modeling and the verification of system properties are thus a key activity. They enable to validate the chosen implementation of the system but they also ease the capitalization on the knowledge of the system. Formalizing the requirements allows to enhance the documentation of the engineering processes by keeping track of design improvements, model refinements, changes in the safety/operational expectations, and so on.

The difficulty of such a V&V approach lies, however, in the fact that, if some techniques and languages exist today to handle system properties, they often involve specific models different from the reference engineering model (i.e. the model commonly used to predict the physical behavior of the system, that is the behavioral model). Such heterogeneity may lead to some errors and thus to flaws in the proof of safety or performance.

The modeling and the checking of system properties concern industries in charge of the design of new

products (e.g. automotive, aerospace) as well as the ones responsible for the operation of long-life products and faced to retrofit due to some material obsolescence and changes in operational constraints (e.g. energy producers). The properties model and the reference engineering model should thus apply for a system involving several physical domains.

For the behavioral model, the increasingly use of the non-proprietary Modelica language [1]-[2] in various industries testifies of the Modelica efficiency to conveniently describe multi-physics behaviors. Besides, thanks to its equation-based and acausal features, the Modelica language appears well-suited to build models reusable and adaptable to the different steps of the engineering cycle.

The objective of this article is thus to study to what extent the properties of a system can be modeled and checked in a Modelica framework.

A similar approach is currently ongoing, within the ITEA2 OpenProd project [3], by linking a Modelica behavioral model to a UML properties model [4]. It actually implies the development of the so-called ModelicaML UML profile [5]-[6]. However the work presented here has been performed within the ITEA2 EuroSysLib project [7] via a collaboration between EDF, Dassault Aviation, Dassault Systèmes and DLR. It takes a different point of view in the sense that the modeling and the checking of system properties are studied in a fully Modelica-based environment. This choice can be explained by a desire to reuse:

- the equation-based feature of Modelica to model properties in a more formal way;
- the same formalism as the one chosen to describe the physical equations of the models in order to ease the expression of their validity domains (which are actually a specific kind of property).

Section 2 clarifies the concept of “property” with no reference made to the way it can be implemented in Modelica. It defines what is a property and sums up the different types of properties. It also specifies the users requirements regarding properties modeling, checking and visualization.

Section 3 aims at formalizing the way a property can be modeled. Like in a formal Property Specification Language [8], the idea is to introduce some theoretical concepts especially useful to express a property in an unambiguous and explicit form. Some notions like “space/time locator”, “state” and “event” are depicted and a list of “operators” to build several types of system properties is given and illustrated on realistic examples.

Section 4 focuses on the technical implementation of these concepts in Modelica. The development of a dedicated library is explained and illustrated on an industrial example taken from the aeronautics domain. The assessment of some system properties is in particular made by simulation using Dymola [9].

Section 5 advocates the extension of the Modelica language in order to improve the applicability range and efficiency of properties modeling for the validation of complex systems.

2 Properties modeling and checking

As mentioned above, the following sub-sections are intended to set up the framework of the study. Independently of the way it can be implemented in Modelica, they are intended to clarify the concept of “modeling and checking properties” and to show its potential use in an industrial context. Notions like a “properties model” or a “behavioral model” are in particular introduced.

2.1 What is a property?

Definition 1: A “property” is an expression that specifies a condition that must hold true at given times and places. It results in a Boolean variable stating whether the property is satisfied or not.

A property may thus specify:

- an **allowed operating domain** the system must not leave for safety reasons;
- an **operational domain** where, for instance, the system operation is optimized for performance;
- the **validity domain of a model** outside of which the corresponding behavioral equations are no longer valid;
- ...

Example: Some realistic properties can be formulated in a textual form such as:

- The power plant should evolve in an allowed (temperature, pressure)-domain;
- Cavitation should never happen in a pump component;
- The characteristics of the pump are only valid for a given range $[Q_1; Q_2]$ of flow rates.
- ...

Different categories of properties may actually be distinguished. A first typology may be drawn depending whether the properties are associated with

the system, a sub-system or a component. But, the properties may also be classified depending on the kind of expectations. Two main kinds of properties may however be highlighted:

- a first kind where the properties characterize the expectations of the designer but also the limitations of the chosen system, sub-systems and components. These properties are expressed independently from any behavioral model;
- a second kind where the properties define some validity domains and are thus attached to specific behavioral models. These properties do not belong to the designer requirements. They only reflect how the designer represents the implementation of its system.

From the tool and language perspective, modeling system and components properties and expressing validity domains are however essentially similar. For the sake of simplicity, the term of “property” will then be used all along the paper to refer to any requirement/limitation the engineer wants to express on its system/sub-system/component or on its model/sub-model.

2.2 Uses of properties modeling

The modeling and the checking of properties may be used in an industrial context to verify and validate each stage of the system development cycle, in particular:

- to enhance the documentation of the system regarding the description of the expected behavior as well as the description of the assumptions made during the modeling of its behavior. This may in particular be useful to ease the capitalization and the transmission of knowledge;
- to improve the engineering processes by expressing the requirements in an explicit and unambiguous form and by keeping track of any evolutions due to design improvements, to changes in operational expectations, to model refinements...

Once the properties have been modeled, a series of tests can then be performed:

- to check the coherence and the completeness of the requirements (e.g. by formal proofs and consistency checks);
- to verify the conformance of the designed system with respect to the initial specifications (e.g. by simulating both the properties model and the behavioral model);
- to validate the Instrumentation & Control (I&C) part of a process on the basis of the

services it should provide to the physical process, during the specification phase, and after the programming phase using hardware-in-the-loop;

- to support advanced modeling approaches like scenarios simulating sequential changes of different operating modes (e.g. simulation of a system entering a dysfunctional mode).

2.3 Distinction between a “behavioral equation” and a “property expression”

Behavioral equations describe a potential implementation of the system at the design phase, or how the system actually works during operation. They are based on physical or empirical laws.

Properties define what the system should guarantee, or in other words what is the validity domain of the system’s behavior. They can also be used to define the validity domain of the model used to represent the system’s behavior.

Example: Let us consider a valve. A behavioral equation can be “ $\Delta P(t) = k / \rho \cdot Q^2(t)$ ” (where $\Delta P(t)$ is the pressure loss across the valve, ρ is the fluid density, k is the pressure loss coefficient of the valve and $Q(t)$ is the mass flow rate through the valve) whereas a property can be “For all t , $\Delta P(t)$ should be greater than ΔP_{\min} to avoid cavitation”.

This distinction is important because behavioral equations and properties are fundamentally different:

- They correspond to **different objectives**: behavioral equations describe **how** the system actually works (e.g. the dynamics of the system) whereas properties define **what** the system should do (e.g. which services it should provide, the prescribed operation domain...);
- They are of **different natures**: behavioral equations define system characteristics which are always localized to a specific part of the system, whereas properties define system characteristics which may be global in time or space, in the sense that they can constrain variables across several periods of time and different locations;
- They involve **different expertise**: writing behavioral equations requires expertise in physical system modeling, whereas defining properties requires expertise in system operation;
- They have **different lifecycles**: the definition of properties occurs during the requirement phase, whereas the modeling of the system’s

behavior occurs during the design phase. For instance, properties may capture the current safety rules, while behavioral equations may describe the current behavior of the system under operation. The impact of the evolution of safety rules on the operation of the system may be assessed by modifying the properties and checking them against the current system's behavior. Inversely, the compliance of system's modifications wrt. the current safety rules can be checked by comparing the modified behavioral equations wrt. the current system's properties.

2.4 Requirements on properties modeling

A property has to be:

- **in interaction with** the behavioral model of the system (since its satisfaction depends on the evolution of the system);
- **transparent** wrt. the dynamic evolution of the system (should not influence the evolution of the system);
- **coherent** with the behavioral model by not implying a too low level of details (properties should not refer to characteristics that are not depicted in the behavioral model);
- **readable** for the sake of documentation and transmission of knowledge;
- **understandable** to ease the interpretation of its potential failure.

The modelling of properties must then be in accordance with these different axioms and an adequate **data model** has to be established in particular to guarantee the transparency of the properties model towards the behavioral model.

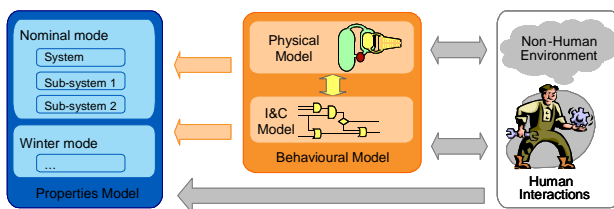


Figure 1: Data model for modeling the system behavior, the evolution of its environment and the properties the system must guarantee

So as to bound the properties model with the behavioral model in such a way that they remain dissociated, we suggest here to physically separate these two kinds of models in two kinds of files. Such a

data model (Figure 1) thus corresponds to a model organized in three different parts:

- the **environmental model** where the characteristics and the evolution of the system environment are specified. This part may in particular be used to set the inputs of the simulation and so to specify some scenarios (e.g. introduction of some component failures, simulation of a series of operator intervention,...)
- the **behavioral model** where the intrinsic characteristics and the evolution of the system are described with behavioral equations. In other words, this part corresponds to the physical modeling of the process and its I&C part;
- the **properties model** where the expected services of the system and the validity domain of the behavioral model are depicted.

In order to ensure the fact that the properties model should be only an observer of the behavioral model, the three parts of this data model must communicate with each other such that:

- the properties model and the behavioral model may access the data described in the environmental model (the properties as well as the behavior of the system may actually change depending on the evolution of the system environment);
- the properties model may access the data described in the behavioral model in order to evaluate whether the dynamic evolution of the physical process and its I&C part stay within the bounds of the prescribed properties domain, but it cannot send any data to influence the behavioral model.

Besides, in order to ease the reading and the construction of the properties model, it may be helpful to organize it into a hierarchy. Depending on the modeler expectations, this hierarchy may be based:

- on the architecture of the studied system;
- on the different states of the studied system and its environment;
- or on a combination of the system architecture and the different states (as in Figure 1).

A hierarchy based on the system architecture may be useful in particular when the architecture of the system changes and the modeler has then to remove or to add some properties related to some specific sub-systems or components. On the other hand, a hierarchy based on the states of the system and of its environment may add further information on how the system should behave (the description of these states

gives in general a better insight into the different operating modes).

In practice, since a different properties model can be built for each operating mode, several properties models can be associated with the same behavioral model.

An advanced data model has thus been imagined to enable the handling of such a situation. As shown in Figure 2, our suggestion is to add a statechart model [10] where the different states of the system and of its sub-systems and components are described. The main idea is then that the statechart model is viewed as a supervisor: it may access the data of the behavioral model, decide in which states the system operates and select the appropriate properties model.

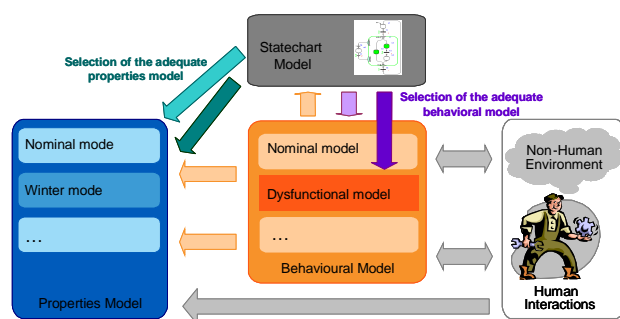


Figure 2: Advanced data model for switching properties models and behavioral models

With the same point of view, it may also be useful to associate several behavioral models with the same system. For instance, if the objective is to anticipate the physical behavior of the system when a fault occurs and to verify whether the corresponding behavior remains in a safe domain, it may be helpful to switch, during the simulation, between a model describing a nominal behavior and another model describing a dysfunctional behavior of the system.

For this particular use, the advanced data model of Figure 2 can be adapted in such a way that the statechart model may:

- access the results obtained from the assessment of the properties to detect if the validity domain of the active behavioral model has been crossed;
- activate, if needed, another behavioral model with an appropriate validity domain.

In such application, let us note however that even if the same statechart model supervises several properties and behavioral models, the hierarchy of the properties may not necessarily correspond to the hierarchy of the behavioral models.

2.5 Requirements concerning the checking, the visualization and the analysis of a property

The question now is to study to what extent the properties and the behavioral models should be coupled together to check whether the properties are satisfied or not.

Two kinds of checks may be imagined: a static check by formal proof and a dynamic check by simulation.

Checks by formal proof can be used to verify the coherence: (1) between the properties themselves (e.g. to verify that properties are compatible between themselves and do not define mistakenly empty operating domains); (2) between the properties and the behavioral model (e.g. to check that the behavioral equations are mathematically compatible with the properties).

Complementarily, **checks by simulation** can be used to verify the properties all along a given scenario such as the “Virtual Verification” method suggested in [11].

Checks by formal proof require that properties and behavioral models are described using **high level formal declarative languages** such as Modelica. Checks by simulation require the **definition of scenarios** with possible occurrences of dysfunctional modes, injection of faults, changes because of human interactions, and so on. Besides, to help the analyst understand the reasons of properties violations, **diagnostics tools** should be provided, such as:

- generation of alarm when a property is violated: a pop-up may appear during the simulation as soon as the non-verification of a property is detected;
- change of component’s visual aspect: the color of a component may change when its corresponding properties are not satisfied;
- edition of a log file: to recap all the properties violations and to signal at first glance when and where the problems have appeared;
- properties filtering: the analyst may need to make a distinction between safety properties and properties indicating some pre-alarms, optimal operating domains, constraints avoiding damages, and so on. The possibility to tag the properties with adequate flags may, for instance, be considered.

This list is however far to be exhaustive, for instance one can also imagine the possibility to introduce some indicators like the probability of a property’s failure.

3 Theoretical concepts used for properties modeling

By definition and similarly to the Behavior Engineering approach [12], a property can be expressed under the generic form: [Where][When][What].

Example: The avoidance of pump cavitation can be expressed by: [In Pump1][for every instant when the pump operates][the fluid pressure at the inlet should be greater than a minimum value].

To meet one of the requirements which is to formalize the expression of the properties, the main idea of the following sub-sections is namely to describe the concepts related to the generic form of a property. In more details, these sub-sections describe what are the “attributes of a property” (i.e. what refer to the where, when and what terms), what they imply (i.e. what “kinds of objects” are used) and how they can be built (i.e. which “operators” are used to construct such attributes).

In order to set up a clear theoretical framework, the following paragraphs are still voluntarily independent of any dedicated language. The implementation in a Modelica environment is studied in Section 4.

3.1 Attributes of a property

As already stated, a property may be expressed under the generic form [Where][When][What].

“Where” is a **space locator** that specifies which part of the system is concerned by the property. The space locator specifies a subset of “everywhere”. It may involve a family of components (e.g. all the pumps), a specific component (e.g. pump n°2), a part of a component (e.g. a specific segment of a pipe), a subset of objects that are in a given state or that are satisfying a given condition (e.g. all the components whose temperature exceeds 240°C). It may also be a subset or a combination of other space locators.

“When” is a **time locator** to indicate at which instants the property has to be satisfied. The time locator is a subset of “always”. It may involve a time instant referring to all the occurrences of a specific event (e.g. when a pump starts), a time period during which a given condition holds true (e.g. as long as the pump operates), a sliding time interval when a given property needs to be satisfied only most of the time (e.g. for no more than 3 minutes over any period of 2 hours). It can also be a combination of several time locators.

“What” refers to the **condition** the system should guarantee (or the assumption the model should satisfy in the case of a validity domain). It consists in an expression that can be evaluated and which results in a Boolean variable stating whether the property is satisfied or not. Because of the variety of properties, conditions can involve physical variables and/or states probed at specific time instants or during specific time periods. They may also imply events, or even a combination of these several kinds of objects with some space and time locators.

3.2 Types of objects implied in the attributes of a property

As shown above, a property may entirely be defined by the association of a space locator, a time locator and a condition to be satisfied. Due to the complexity of the systems and the different types of properties the designer is interested in, these attributes have to deal with numerous kinds of objects such as:

- **instances of models** (e.g. Pump1, SensorMT018...);
- **geometric data** (e.g. segment[0.2...0.8] of Pipe3);
- **physical variables** and/or parameters of different physical types;
- **states** of the system, sub-systems and components (e.g. since the expected services are often depending on the different operating modes, a property may concern a component only when it is not in a dysfunctional state);
- **events** that characterize external stimuli of the system (e.g. human intervention, evolution of the system’s environment) or internal changes of the sub-systems and their components (e.g. fault during a valve opening);
- combinations of instances of models, geometric details, physical variables, states and events built thanks to some specific **operators** (e.g. Pump1 and Pump2, for every instant where Pump1 operates, all pumps except Pump3...).

The two following paragraphs define in more details what we mean by the notions of “state” and “event” which may be less naturally intuitive. The concept of “operator” to build adequate properties attributes is then studied in Section 3.3.

3.2.1 State

Definition 2: A “state” is a discrete variable that characterizes an aspect of a system, a sub-system or a component. It can take its values only within a fi-

nite set of enumerated values. It is defined according to a specific point of view on the system.

Example: States may correspond to different operating modes (e.g. maintenance/normal operation), operating conditions (e.g. cold winter/hot summer), physical behaviors (e.g. pump/turbine mode)...

A state may have one (or several) attached sub-state variable(s) and the same system, sub-system or component may have multiple independent states.

As explained above, the notion of state can also help to organize the properties and the behavioral models into a hierarchy.

3.2.2 Event

Definition 3: An “event” is an object that is generated at a given time instant to signal the occurrence of a fact. It carries at least two pieces of information: the date and the class of the event. The former indicates when the change has occurred while the latter defines what has happened. An event has no duration and does not characterize what are the consequences of the change on the system behavior.

Example: The starting of a motor may generate an event.

In some cases, sub-classes of the event concept may be created to provide further information such as a probability distribution, a frequency of appearance, and so on. A distinction may also be made depending on the location of the change. For instance, internal events are related to the evolution of some variables in the behavioral models while external events correspond to changes in the system’s environment.

The introduction of the event concept can especially be used to define some simulation scenarios with injections of faults, control or perturbation actions.

3.3 Operators to build property attributes

An operator is defined here as a function that constructs an object as output given one or several objects as inputs. Inputs and outputs may be of the same type, or of different types. Operators are of prime importance to build space/time locators or even conditions:

- when a simple observation of the variables available in the behavioral model is not sufficient to describe what the system should guarantee or when the model is valid (e.g. when a behavioral model involves only a mass flow rate variable and the corresponding property is expressed in terms of volume flow

rate, an operator has to be used to perform the unit conversion);

- or when it is easier to express it as a function or a combination of other attributes.

Operators may be classified depending on the types of their inputs and outputs. From the analysis of industrial needs based on EDF and Dassault-Aviation use-cases, some operators have been identified as particularly useful, such as:

- **arithmetical operators** and usual functions: +, -, *, ÷, cosinus, absolute value,...;
- **logical operators:** and, or, ...;
- **set operators:** all, in, ∈, ∉ (creation of a set), ∪ (sets union), ∩ (sets intersection), $card()$ (cardinal of a set), \bar{S} (complement of a set), \ (subtraction of a subset), ...;
- **operators on time and events:** for, when, while, always, never, delay between two events, duration of a time period, count of the number of events, events synchronization;
- **dedicated operators:** >, <, ≥, ≤ (thresholds), ⊂, ⊄ (domain inclusion), Δ (ramp), A (accumulation), $freq()$ (frequency).

Among this (non-exhaustive) list, some operators refer to well-known concepts in mathematics or computer programming, while other correspond to operators more specific to the modeling of properties for physical systems. The aim of the following sections is to give a better insight into these dedicated operators by furnishing their mathematical description and illustrating their uses. Their implementation in Modelica will then be further discussed in Section 4.

3.3.1 Threshold operator

Definition 4: The “threshold operator” defines a lower (or an upper) limit that a variable should not exceed.

This operator may be used to build a time locator or a condition.

Examples: In the case of an heat exchanger, an external leakage may appear if the pressure and the temperature both exceed given maximum values. An internal leakage may also occur if the number of cycles is superior to a specific limit. Air bearing can be destructed if its rotational speed crosses a maximum value. If the Mach number is superior to a specific limit, the model of an air pipe may predict pressure losses with less confidence.

satisfied) which are built through specific operators from objects provided by the behavioral model(s).

The following examples serve as an illustration on how a property expressed in a textual form may be reformulated in a formal manner. Through some combination processes, they also show how complex properties may be entirely described from the short list of formal concepts previously described.

Examples: To avoid the Pump1 cavitation, the pressure at the inlet should be greater than a minimum value \rightarrow [In Pump1] [for all t | state = operating] [condition $P(t) > P_{min}$].

To ensure the performance of the cooling system, at least two pumps should be operating \rightarrow [In Cooling-System] [Always] [condition cardinal (set (Pump | state = operating)) ≥ 2].

To avoid the turbine wheel erosion, no liquid water should enter at the inlet during more than 30 minutes \rightarrow [In Turbine][Always][condition duration (Inlet-Water.state = liquid) ≤ 30 minutes].

4 Modeling of properties in a Modelica framework

First, a Modelica library dedicated to the modeling of properties that has been developed during the EuroSysLib project is presented and illustrated on an industrial use-case. Then, the current limitations of this library are discussed. Finally, a rationale for a Modelica language extension to support properties checking by formal proof is given.

4.1 Modeling of properties with a dedicated Modelica library

4.1.1 Purpose of the library

The aim of the library is to make checks on parts of an architecture defined by a Modelica behavioral model. Its particular features are the following:

- It enables checks during simulations, gets and stores information in case of detected defect for actions (e.g. stops the simulation and starts the next one according to specified criteria);
- It enables reuse of the properties by parameterizing them according to the potential uses of the model (e.g. mission profile, specific boundary or environmental conditions...), stores the properties in a catalog for reuse;
- It enables dysfunctional analysis: check of properties must not influence model simulation (e.g. potential change of time step com-

ing from properties evaluation must not lead to an unwanted decrease of results accuracy). But, properties could be used to change the behavior of models with defect (detected by properties observers). In this case models should be modeled with different behaviors which could be activated on demand (with currently smooth change between behaviors due to the change of the equation structure).

The major particularity is that checks are not done as post-processing, but on the fly at run-time.

4.1.2 Use-Case: Environmental Control System (ECS)

The simple ECS used for testing properties is defined by two main parts: (1) the Cold Air Unit (CAU), made of pipes, heat exchangers, compressors and turbines, and which controls air characteristics provided to the cabin and bays; (2) the bleed, which provides air from the engines to the CAU (Figure 5).

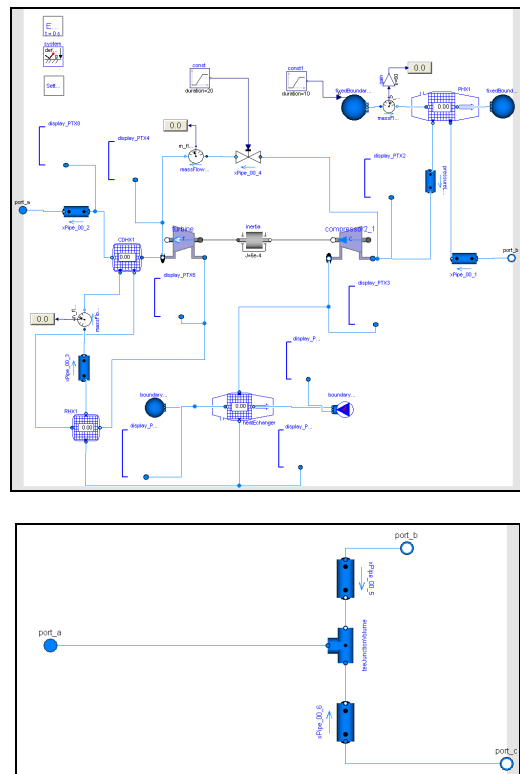


Figure 5: Cold Air Unit and Bleed of the Environmental Control System

The ECS must be compliant with many requirements. These requirements are classified according to 6 categories:

1. **threshold monitoring**, which deals with crossing of threshold (see paragraph 3.3.1 for examples of properties on the heat exchanger, air bearing and pipes);

2. **operating domain monitoring**, which mainly deals with conditions regarding the location of couple of values (or more) inside a defined area or inside a volume. Currently only requirements regarding 2D area is defined here, but conditions with more than two dimensions could occur (see paragraph 3.3.2 for example of allowed domains for a compressor);
3. **rate of change monitoring**, which deals with conditions with derivatives (see paragraph 3.3.3 for criteria concerning the cabin altitude rate of change);
4. **accumulation monitoring**, which deals with conditions with integration (see paragraph 3.3.4 for example of a condition to prevent the clogging of the exchanger);
5. **oscillation monitoring**, which deals with conditions based on oscillation characterizations (occurrences, frequencies) (see paragraph 3.3.5 for the example of a regulating valve);
6. **monitoring with space/time locators**, which test conditions linked to location of components or events (see paragraph 3.4 for the example of a property to avoid turbine wheel erosion).

4.1.3 Structure of the library

The presented library is currently dedicated to the ECS use-case (with behavioral components developed by Dassault-Aviation) and properties observers. This use-case appears here as a library divided into two main parts (Figure 6):

1. A generic part, called “PropertyObservation”. It contains:
 - a. examples, especially models from DLR, which propose two ways (a direct link or a bus) for connecting the properties to the ECS model.
 - b. the ECS model using the second type of properties connection since it appears as the most generalized and readable way for complex systems involving numerous properties.
2. A second part which contains use-cases. It is split into models and requirements:
 - a. The models are here focused on an ECS system simulated with dry or moist air;
 - b. The properties are a collection of specific properties built as generically as

possible and classified into several types of properties.

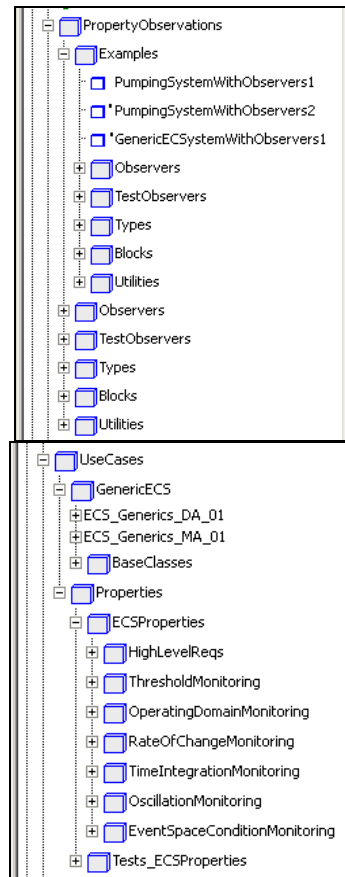


Figure 6: The two main parts of the library

4.1.4 Process for properties modeling and checking

4.1.4.1 Connecting models to properties

When a system must be checked regarding its compliance with requirements, a good process is to integrate the model inside a virtual test bench by extending it. In this way the model can be checked according to several sets of requirements.

An expandable bus called RequirementBus is added to the models by drag and drop from the library. Requirements or sets of requirements are then connected to the RequirementBus. Parameters of the requirements can be adjusted according to the analysis. All values which must be provided by the model are then automatically available on the bus.

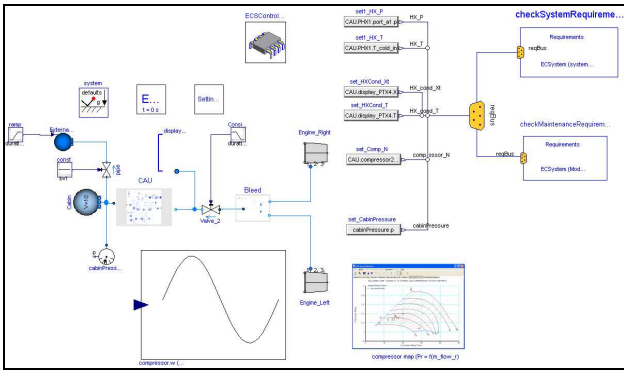


Figure 7: ECS use-case

All the necessary interfaces must then be defined one by one using components called Modelica.Blocks.Sources.RealExpression from the Modelica Standard Library.

The component RealExpression must be linked to a variable in the model as in Figure 8 for the cabin pressure.

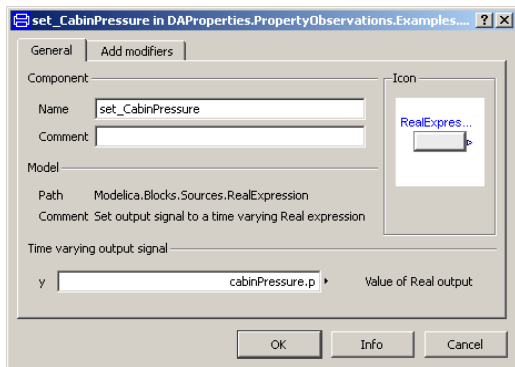


Figure 8: Example of value of a component RealExpression

When connecting the component, a window appears for mapping variable selections (Figure 9). It allows the user to select which variable must be mapped to the component RealExpression.

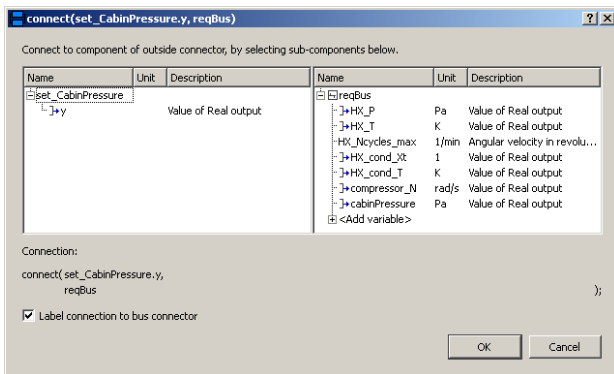


Figure 9: Window appearing for connecting RealExpression to input variable within Requirements

4.1.4.2 Hierarchical decomposition of properties

Requirements may be complex with many elements. Therefore putting all requirements at the same level may be cumbersome.

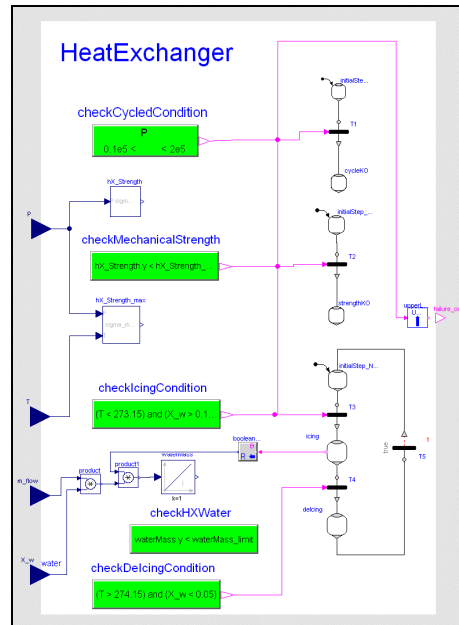


Figure 10: Requirements for a heat exchanger

A simple example on the ECS heat exchanger is shown in Figure 10 where the properties are composed of three main conditions:

- **checkCycledCondition:** counts the number of pressure cycles seen by the heat exchanger and sets a warning when this number is upper a threshold;
- **checkMechanicalStrength:** computes an equivalent stress within the heat exchanger and compares it to an allowed maximum stress;
- **checkIcingCondition, checkDeIcingCondition and checkHXWater:** check icing and deicing conditions, and the amount of water inside the heat exchanger. Typically, if the mass of water is above a limit, the heat exchanger could be partially clogged and the simulation could be not valid if the behavioral model is not adapted to this particular situation. When attaining this operating condition, it is interesting to continue the simulation with the properly modified behavioral model to analyze the consequences of being outside of the nominal domain (dysfunctional analysis).

These requirements stand for the heat exchanger but all the types of properties defined in Section 3 have been investigated within the complete ECS use-case.

In fact, many other properties observers could be added and if we consider observers of other components or sub-systems it seems to be cumbersome to put them all in the same view. Therefore the library has been enhanced to support hierarchical decomposition of properties. In particular a component called UpperLevel has been introduced to transmit the result as an OR function of its inputs.

4.1.5 Unit test for a heat exchanger

Properties components have been tested with specified inputs to check that their behaviors were correct. Figure 11 shows different properties states for a heat exchanger.

During simulation, the visual indicator stays green as long as no defect is detected (state 1). When a defect occurs, the edge of the indicator turns to red (state 2) and goes back to green as soon as the defect detection disappears. To keep the memory of a defect detected during the simulation, another outside red square is added and remains until the end of the simulation (state 3).

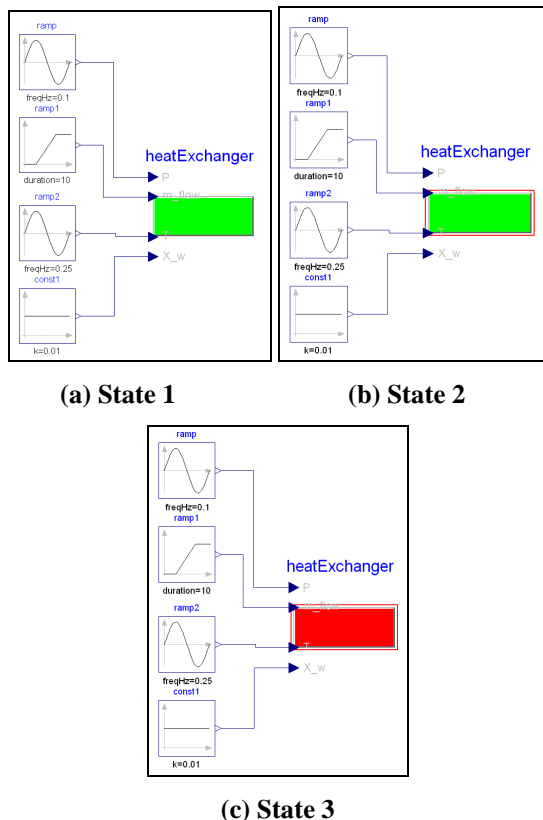


Figure 11: Warning indicators of a set of properties

For a detailed analysis, it is possible to access the internal warning indicators of each property as shown in Figure 12.

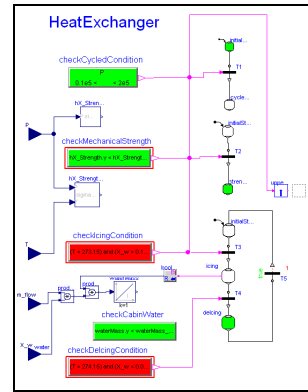


Figure 12: Internal warning obtained at the lower level of requirements (detailed level)

It is also possible to investigate more deeply what has happened by plotting all variables of interest.

5 Formal modeling of properties with Modelica language extensions

The previous section has shown that the development of a dedicated Modelica library is efficient to model the main properties implied in the ECS industrial use-case. Two current limitations have however to be mentioned.

Firstly, even simple properties cannot be modeled as soon as they imply space or time locators.

Examples : Currently the following properties cannot be modeled.

→ [In Turbine] [Always] [condition duration (InletWater.state = liquid) ≤ 30 minutes]

→ [In CoolingSystem] [Always] [condition cardinal (set (Pump | state = operating)) ≥ 2].

Secondly, even if the properties library features the all main operators needed to model properties, it only supports the construction of the properties in a block-diagram way: many components must be connected to form one simple property. This approach is quite in contradiction with the Modelica spirit as it emphasizes a graphical modeling approach over a formal equation modeling approach. Therefore, it does not comply with the fundamental requirements for the formal description of properties. This leads to several potential limitations, such as the impossibility to check properties by static proofs, or the impractical association of validity domains to behavioural equations.

As for the modeling of the physical behaviors, a formal (i.e. an equation-based) approach presents numerous advantages to model properties (Figure 13).

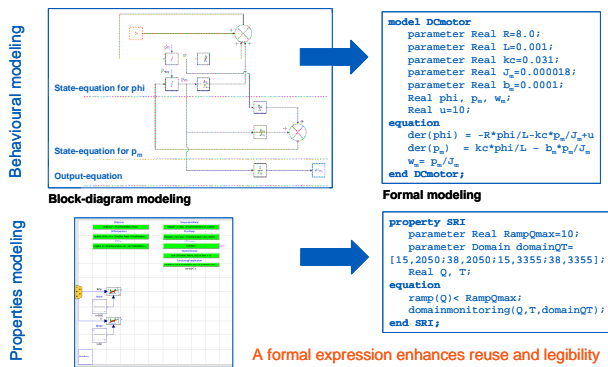


Figure 13: Advantage of a formal (equation-based) approach for behavioral and properties models

A formal description allows to:

- provide explicit and unambiguous specification of properties and thus avoid some potential misunderstandings and mistakes;
- enhance the legibility and so the reuse of the properties models;
- improve the test coverage by automating the checking procedures;
- enable some static tests (i.e. tests performed without any simulation) on the coherence and the completeness of the properties.
- associate validity domains to behavioural equations, and perform various checks on them.

Examples: For a model where the two properties “[In Pump1][Always][condition $P(t) > P_{min}$]” (to avoid pump cavitation) and “[Always][condition $P(t) < P_{max}$]” (to guarantee the flow direction of radioactive leaks) should be satisfied, a first check should ensure that there is no contradiction between the numerical values of P_{min} and P_{max} .

In another model, if the two following properties “[For $0\text{ °C} < T < 15\text{ °C}$][condition ...]” and “[For $22\text{ °C} < T < 38\text{ °C}$][condition ...]” should be fulfilled, one may wonder if there is some incompleteness in the requirements and what should happen when the temperature is between 15 °C and 22 °C .

Hence modeling the properties with an equation-based approach will give the possibility to perform formal transformations and verifications on both the properties and behavioral models. This will contribute greatly to improve system validation by increasing the coverage and the rigor of the verifications.

The use of Modelica for that purpose may only be done by introducing natively in the language the concepts of space/time locators and dedicated operators.

6 Conclusions

In order to improve the V&V process, this article deals with the modeling and checking of system properties. The study is made within a fully Modelica-based framework and encompasses with the term “property” the modelling of any requirement/limitation the engineer wants to express on its system/subsystem/component or on its model/sub-model.

Imagined as complementary to the ModelicaML approach, modelling system properties directly in Modelica is justified here as a desire to: (1) use an equation-based language to express the properties in an unambiguous way; (2) choose a formalism closed to the one used for expressing the physical equations in order to ease the formulation of the validity domains of the models.

After having introduced some theoretical concepts to formally describe a property, some requirements have been listed on how the properties and the behavioural models should communicate to check virtually whether the properties are satisfied or not.

The development of a Modelica library dedicated to the modelling of properties has then been explained and illustrated on an industrial example taken from the aeronautics domain. Even if several operators have been especially built to cover the most types of properties, two current limitations have however to be raised: (1) even simple properties cannot be modeled as soon as they imply some space or time locators; (2) the properties are actually modeled in a block-diagram way which is inconsistent with the ambition of performing formal proofs.

Further work has then to be investigated to make up for these aspects and concrete proposals should be made to introduce natively in the Modelica language the concepts of space/time locators and dedicated operators.

Acknowledgements

This work was partially supported by the pan-European ITEA2 program and the French government through the EuroSysLib project.

References

- [1] Information available on the Modelica Association web site: <http://www.modelica.org>

- [2] P. Fritzson, *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*, Wiley IEE Press, 944 pages, February 2004.
- [3] Information available on the OpenProd web site: <http://www.ida.liu.se/~pelab/OpenProd/>
- [4] Information available on the official UML web site: <http://www.uml.org>
- [5] W. Schamai, P. Fritzson, C. Paredis, A. Pop, *Towards Unified System Modeling and Simulation with ModelicaML: Modeling of Executable Behavior Using Graphical Notations*, in Proceedings of the 7th Modelica Conference, Como, Italy, September 20-22, 2009.
- [6] W. Schamai, *Modelica Modeling Language (ModelicaML): A UML Profile for Modelica*, technical report in Computer and Information Science, n° 2009:5, Linköping University Electronic Press, 49 pages, 2009.
- [7] Information available on the EuroSysLib web site: <http://www.eurosyslib.com>
- [8] *Property Specification Language – Reference Manual*, Accellera technical report, USA, June 2004.
- [9] Dymola software, Dassault Systèmes, information available at: <http://www.dymola.com>
- [10] D. Harel, *Statecharts: A visual formalism for complex systems*, in Science of Computer Programming, 8(3):231-274, June 1987.
- [11] W. Schamai, P. Helle, P. Fritzson, C. Paredis, *Virtual Verification of Systems Design against System Requirements – A Method Proposal*, in Proceedings of the 3rd International Workshop on Model Based Architecturing and Construction of Embedded Systems (ACES 2010), in conjunction with MODELS 2010, Oslo, Norway, October 4, 2010.
- [12] T. Myers, P. Fritzson, R.G. Dromey, *Seamlessly Integrating Software & Hardware Modelling for Large-Scale Systems*, in Proceedings of the 2nd International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools (EOOLT 2008), Paphos, Cyprus, July 8, 2008.
- [13] *Eurosyslib sWP7.1 DGT116083B Dysfunctional Use Cases and User Requirements*, 2010.
- [14] *EuroSysLib sWP7.1 DGT124618 Properties Evaluation Report*, 2010.
- [15] *EuroSysLib sWP7.1 Properties Modeling*, 2010.

Dynamic modeling of a solid oxide fuel cell system in Modelica

Daniel Andersson Erik Åberg Jonas Eborn
 Modelon AB, Ideon Science Park, SE-223 70 Lund, Sweden

Jinliang Yuan Bengt Sundén
 Department of Energy Science, Lund University, Box 118, SE-221 00

Abstract

In this study a dynamic model of a solid oxide fuel cell (SOFC) system has been developed. The work has been conducted in a cooperation between the Department of Energy Sciences, Lund University, and Modelon AB using the Modelica language and the Dymola modeling and simulation tool. The objective of the study is to investigate the suitability of the Modelica language for dynamic fuel cell system modeling.

Fuel cell system modeling requires a flexible modeling tool that can handle electronics, chemistry, thermodynamics and the interaction between these. The core of the fuel cell is the electrolyte and the electrodes. The cell voltage generated depends on the fluid molar compositions in the anode and cathode channels. The internal resistance varies depending on several cell properties. The electrical current through the cell varies over the cell area and is coupled to the rate of the chemical reactions taking place on the electrode surface. Other parts of the system that is also included in the model are pre-processing of the fuel, combustion of the fuel remaining after passing through the cell and heat recovery from the exhaust gas.

A cell electrolyte model including ohmic, activation and concentration irreversibilities is implemented and verified against simulations and experimental data presented in the open literature. A 1D solid oxide fuel cell model is created by integrating the electrolyte model and a 1D fuel flow model, which includes dynamic internal steam reforming of methane and water-gas shift reactions. Several cells are then placed with parallel flow paths and connected thermally and electrically in series. By introducing a manifold pressure drop, a stack model is created. This stack model is applied in a complete fuel cell system model including an autothermal reformer, a catalytic afterburner, a steam generator and heat exchangers. Four reactions are modeled in the autothermal reformer; two types of methane steam

reforming, the water-gas shift reaction and total combustion of methane. Several simulations of systems and individual components have been performed, and when possible been compared with results in the literature. It can be concluded that the models are accurate and that Dymola and Modelica are tools well suited for simulations of the observed transient fuel cell system behaviour.

Keywords: SOFC; fuel cell; system model; dynamic reaction; reforming

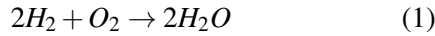
Nomenclature

ASR	Area specific resistance
C	Cross-plane resistance area
D	Diffusion coefficient
E	Ohmic symmetry factor
F	Faraday constant
G	Gibbs free energy
H	Specific Enthalpy
i	Current density
i_0	Exchange current density
J	Non-dimensional strip width
L	Characteristic length
n_e	Number of exchanged electrons
p	Partial pressure
P	Pressure
R	Universal gas constant
r	Reaction rate
t	Thickness
T	Temperature
V	Voltage
X	Cell pitch length
y	Molar fraction

- α Charge transfer coefficient
- β Parameter in Eq. (10)
- γ Pre-exponential factor
- η Voltage loss
- ν Effectiveness factor
- ρ Resistivity
- σ Conductivity
- Ω Denominator in reaction kinetics

1 Introduction

A solid oxide fuel cell is an electrochemical device that produces energy by oxidizing fuel from a replenishable source. A number of fuels are usable in SOFCs, thanks to internal reformation of the fuel. The overall reaction releasing energy is oxidation of hydrogen



and thus the only byproduct produced aside from heat is water. The working principle is that oxygen is ionized at the cathode electrode by electrons from an external electrical circuit. The oxygen ions diffuse through the electrolyte to the anode electrode where they react with hydrogen gas, forming steam.

The solid oxide fuel cell is a high temperature fuel cell, most oftenly operating at 750 - 800°C. The high temperature removes the need for expensive platinum catalyst and allows gases such as natural gas to be used as fuel directly. SOFCs are suitable for many types of system configurations. These include all sizes of combined heat and power applications, where a stationary fuel cell system is used to provide both electricity and heat to buildings, etc. The power generation in such systems can range from 2 kilowatts to several megawatts. They can also be used as auxiliary power units, or in hybrid systems where a fuel cell is combined with a gas turbine [1, 2, 3]. For all types of fuel cells it is important to keep track of the fuel composition going into the cell, as this affects the performance of the cell and in some cases can damage the cell. In this study a model of a SOFC system fueled with natural gas has been developed using Modelica and Dymola. Hydrogen is obtained from the natural gas via reformation. The fuel is reformed in an autothermal reformer prior to entering the cell, as well as during the flow through the cell channel; a process known as internal reforming. Heat is recovered from the exhaust gas and is used to generate steam from liquid water, as well as pre-heating the natural gas and air supplies.

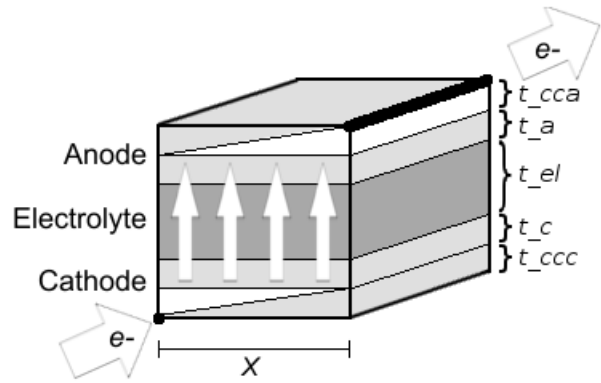


Figure 1: Geometry of the cell for calculation of the ohmic resistance

2 Single cell model

The ideal open circuit cell voltage is given by the Nernst equation [4]:

$$V_{Nernst} = -\frac{\Delta G}{2F} - \frac{RT}{2F} \ln \left(\frac{p_{H_2} p_{ref}^{0.5}}{p_{H_2} p_{O_2}^{0.5}} \right) \quad (2)$$

where ΔG is Gibbs free energy from the reaction and p_{ref} is the standard pressure 0.1 MPa. When a current is applied, the cell voltage drops due to ohmic, activation and concentration losses, and thus the total voltage over the cell can be expressed as

$$V = V_{Nernst} - \eta_{Ohmic} - \eta_{act} - \eta_{conc} \quad (3)$$

2.1 Ohmic loss model

The calculation of the ohmic loss is based on the formulas for ohmic resistance of a cell with diagonal terminals, which is based on the integrated planar cell geometry as shown in Figure 1 [5]. For this geometry the area specific resistance is given by

$$ASR = CJ \left[\coth(J) + B \left(J - \tanh \left(\frac{1}{2}J \right) \right) \right] \quad (4)$$

where C is the cross-plane resistance area, given by:

$$C = \rho_{ct} t_c + \rho_{ccc} t_{ccc} + \rho_{el} t_{el} + \rho_{at} t_a + \rho_{cca} t_{cca} \quad (5)$$

B is given by

$$B = \frac{E}{(1+E)^2} \quad (6)$$

$$E = \left(\frac{t_{cca}}{\rho_{cca}} + \frac{t_a}{\rho_a} \right)^{-1} \left(\frac{t_{ccc}}{\rho_{ccc}} + \frac{t_c}{\rho_c} \right) \quad (7)$$

where E is the ohmic symmetry parameter. J is given by

$$J = \frac{X}{L} \quad (8)$$

$$L = \sqrt{\frac{\rho_{el} t_{el}}{\left(\frac{t_{cca}}{\rho_{cca}} + \frac{t_a}{\rho_a}\right)^{-1} + \left(\frac{t_{ccc}}{\rho_{ccc}} + \frac{t_c}{\rho_c}\right)^{-1}}} \quad (9)$$

where X is the cell pitch length as indicated in Figure 1 and L is the characteristic length [5]. The ohmic conductivities of the cell materials are temperature dependent, and commonly (e.g. [5]) calculated according to:

$$\sigma = \rho^{-1} = \beta_1 \exp\left(-\frac{\beta_2}{T}\right) \quad (10)$$

The voltage drop is then calculated from the cell current density and the area specific resistance according to:

$$\eta_{Ohmic} = ASR \cdot i \quad (11)$$

2.2 Activation loss model

Activation losses occur at both the anode and cathode and derives from the fact that the reacting species in chemical reactions must overcome an energy barrier, i.e. the activation energy of the reaction. The activation loss shows a nonlinear current dependency related to a parameter known as the *exchange current density*, i_0 . For current densities $i < i_0$ the activation loss is low, typically in the order of 0.01 V, and for current densities $i > i_0$ the activation loss grows according to the Tafel equation:

$$\eta_{act} = \frac{RT}{2\alpha F} \ln\left(\frac{i}{i_0}\right) \quad (12)$$

For current densities $i < i_0$ the activation loss is assumed to be proportional to the logarithm of the current density on the form:

$$\eta_{act} = k \cdot \ln i \quad (13)$$

Activation losses occur at both the anode and cathode sides which have different exchange current densities, denoted $i_{0,a}$ and $i_{0,c}$ respectively. These are calculated from the Arrhenius law and the composition of the reacting gases according to [5]:

$$i_{0,c} = \gamma_c \left(\frac{p_{O_2}}{p_{ref}}\right)^{0.25} \exp\left(-\frac{E_{act,c}}{RT}\right) \quad (14)$$

$$i_{0,a} = \gamma_a \left(\frac{p_{H_2}}{p_{ref}}\right) \left(\frac{p_{H_2O}}{p_{ref}}\right) \exp\left(-\frac{E_{act,a}}{RT}\right) \quad (15)$$

2.3 Concentration loss model

Concentration losses accounts for the fact that when reactions occur, reactants and products must diffuse from the bulk flow to the reaction sites (and vice versa), through the porous electrodes. Because of this the actual pressure of the reactants and products differ from those in the bulk flow, and the voltage over the cell decreases. Typically this voltage drop is very low until the current density reaches a limiting current. Above this limit the concentration loss will have severe impact on cell performance and life time. The voltage drop is expressed as a function of the molar fractions y_i of the gases at the reaction sites as [5]:

$$\eta_{conc} = -\frac{RT}{n_e F} \left[\ln\left(\frac{y_{H_2}^* y_{H_2O}^O}{y_{H_2}^O y_{H_2O}^*}\right) + \frac{1}{2} \ln\left(\frac{y_{O_2}^*}{y_{O_2}^O}\right) \right] \quad (16)$$

where y^* indicate molar fraction at the reaction site, and y^O indicate molar fraction in the bulk flow. The molar fractions at the reaction sites are calculated from those in the bulk, the current density through the electrolyte, the thickness of the anode and cathode, the bulk pressure and the diffusion coefficients [5], according to:

$$y_{O_2}^* = 1 + (y_{O_2}^O - 1) \exp\left(\frac{iRTt_c}{4n_e F D_{O_2} P}\right) \quad (17)$$

$$y_{H_2}^* = y_{H_2}^O - \frac{iRTt_a}{2n_e F D_{H_2} P} \quad (18)$$

$$y_{H_2O}^* = y_{H_2O}^O + \frac{iRTt_a}{2n_e F D_{H_2O} P} \quad (19)$$

2.4 Simplified cell model

In addition to the complete polarization model, modelling the internal losses physically as described above, also a simplified cell model was developed. In this model the internal losses are approximated by the following empirical correlation for area specific resistance [4]:

$$ASR(T) = ASR_0 \exp\left(\frac{E_a}{R} \left(\frac{1}{T} - \frac{1}{T_0}\right)\right) \quad (20)$$

where the constant ASR_0 is the area specific resistance at temperature T_0 . This simple model is not as accurate as the complete model, but gives shorter simulation times.

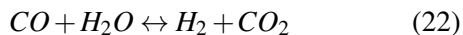
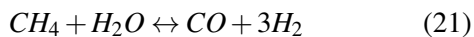
2.5 Implementation

The cell models are based on a template model containing common parts in all cell models. This includes

connectors for mass flow, heat transfer and electrical current. Also a temperature state is introduced and energy balance is defined. The fuel and air compositions have large variations along the flow path due to reactions. To take this into account the cell is discretized in the fuel flow direction. The cell model is prepared for this by vectorizing all connectors and states that vary along the flow path. There is no heat transfer directly between the temperature states, this is covered by heat transfer to the surrounding wall and fluids, which also have heat transfer to the surrounding temperature states. The cell current is also discretized in the same way. For each element the current is coupled to the mass flow rate, and thereby the reaction rate at the electrodes, according to Faraday's laws of electrolysis.

3 Substack model

The cell model is connected to flow channel models in what we call a substack model. The flow channel models are volume models, discretized in flow direction. Every discrete element has a unique fluid composition and energy content. The anode channel include reactions for steam reforming of methane (21) and water-gas shift (22).



In the channel model these reactions are characterized by a reaction object which contains the stoichiometry matrix and calculates the equilibrium constants of the reactions. The equilibrium constant is calculated from Gibbs free energy of the reaction, which is evaluated from the medium model, and the partial pressures of the reactants. The actual reaction rate is then calculated in the channel model from the deviation of the fluid composition from the calculated equilibrium. Medium properties are calculated by medium models from Modelon's CombiPlant library which is a model library for combined cycle power plant simulation. These medium models are compatible with the ones in the Modelica Standard Library, but also includes a model `ReactionProperties` which calculates properties related to reactions, such as Gibb's free energy due to reaction and reaction constants.

The two channel models are connected to a cell model according to Figure 2. The number of discrete elements are equal in the channel models and the cell

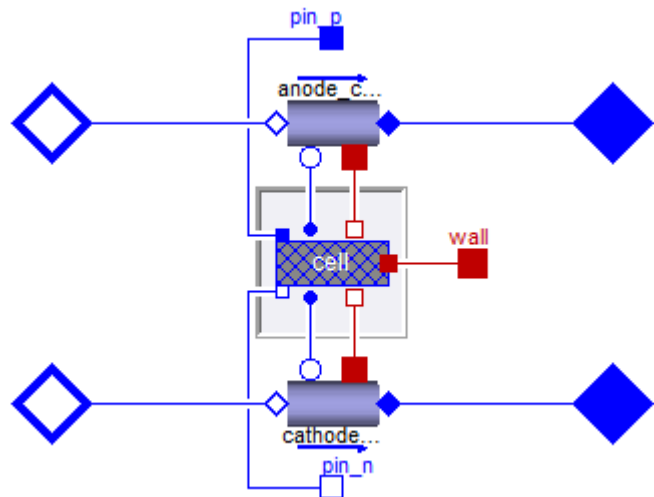


Figure 2: Graphical layout of the substack model

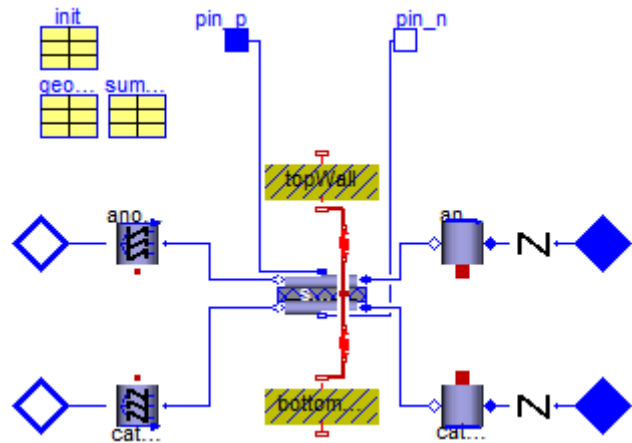


Figure 3: Graphical layout of the stack model

model. Thus the connections for mass and heat transfer are done per element, and each element in the cell model has a unique fuel and air state to evaluate. The substack models any number of cells by multiplying the affected quantities with the number of cells. The purpose of the substack model is to model a number of cells in the stack which can be assumed to have the same operating conditions.

4 Stack model

The complete stack is modeled by connecting several substack models, electrically and thermally in series, as shown in Figure 3. This model consists of a vector of substack models, inlet manifold volumes with pressure drop, and outlet manifold volumes. The first and last substacks in the vector are also thermally connected to thermal masses, modeling the heat capacity of the metal casing. The pressure drops in the in-

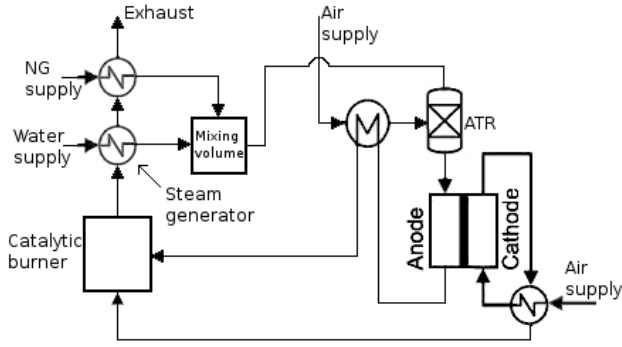


Figure 4: Layout of the implemented fuel cell system

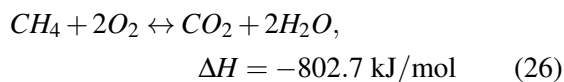
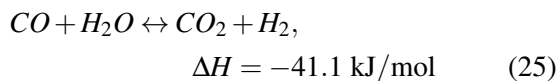
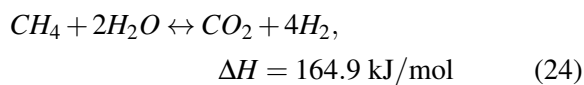
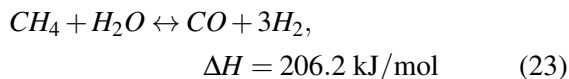
let manifold volumes generates a unique pressure at each substack inlet, corresponding to a U-type manifold form. The stack model thereby has 3D effects from the heat boundary conditions and flow distribution.

5 System model

The system configuration is based on that presented in [6], with some modifications. The system layout, presented in Figure 4, includes an autothermal reformer (ATR), a steam generator and a catalytic burner. The outlet gases from the stack are fed through heat exchangers in order to pre-heat air and is then fed to a catalytic burner. The hot burner exhaust gas is used to vaporize liquid water to steam, which is needed for the reforming reactions in the ATR and stack and to pre-heat the natural gas from ambient temperature.

5.1 ATR unit

A large number of reactions can occur in an ATR unit, but in order to reduce the number of calculations only the most significant reactions are considered; steam reforming (23), (24), the water-gas shift reaction (25) and total combustion of methane (26) [7]:



The presented values for ΔH of the reactions are valid at 298 K. These values are not used explicitly in the model, but are presented to show the mix of endothermic and exothermic reactions taking place in the ATR unit, allowing it to operate without any need for external heating or cooling systems. In our model enthalpies are evaluated from the medium model and depends on the thermodynamic state and composition of the gas in the ATR unit. The model is implemented as a volume model with dynamic reactions. The reaction rates are calculated as [7]:

$$R_1 = \frac{k_1}{p_{\text{H}_2}^{2.5}} (p_{\text{CH}_4} p_{\text{H}_2\text{O}} - \frac{p_{\text{H}_2}^3 p_{\text{CO}}}{K_I}) \times \frac{1}{\Omega^2} \quad (27)$$

$$R_2 = \frac{k_2}{p_{\text{H}_2}^{3.5}} (p_{\text{CH}_4} p_{\text{H}_2\text{O}}^2 - \frac{p_{\text{H}_2}^4 p_{\text{CO}_2}}{K_{II}}) \times \frac{1}{\Omega^2} \quad (28)$$

$$R_3 = \frac{k_3}{p_{\text{H}_2}} (p_{\text{CO}} p_{\text{H}_2\text{O}} - \frac{p_{\text{H}_2} p_{\text{CO}_2}}{K_{III}}) \times \frac{1}{\Omega^2} \quad (29)$$

$$R_4 = \frac{k_{4a} p_{\text{CH}_4} p_{\text{O}_2}}{(1 + K_{\text{CH}_4}^C p_{\text{CH}_4} + K_{\text{O}_2}^C p_{\text{O}_2})^2} + \frac{k_{4b} p_{\text{CH}_4} p_{\text{O}_2}}{1 + K_{\text{CH}_4}^C p_{\text{CH}_4} + K_{\text{O}_2}^C p_{\text{O}_2}} \quad (30)$$

$$\Omega = 1 + K_{\text{CO}} p_{\text{CO}} + K_{\text{H}_2} p_{\text{H}_2} + K_{\text{CH}_4} p_{\text{CH}_4} + K_{\text{H}_2\text{O}} \frac{p_{\text{H}_2\text{O}}}{p_{\text{H}_2}} \quad (31)$$

Here R_1 , R_2 , R_3 and R_4 are the corresponding reaction rates to reactions (23), (24), (25) and (26), respectively. k_j is the Arrhenius reaction constant for reactions $j = 1, \dots, 4$ and is calculated using parameters from literature [7]. K_j is the equilibrium constant for reaction j and is listed in Table 1. The Van't Hoff species adsorption constants K_i and K_i^C for species i are calculated in a similar way according to [7]:

$$K_i = K_{oi} \times \exp\left(\frac{-\Delta H_i}{RT}\right) \quad (32)$$

$$K_i^C = K_{oi}^C \times \exp\left(\frac{-\Delta H_i^C}{RT}\right) \quad (33)$$

Reaction, j	Equilibrium constant, K_j
1	$K_I = \exp\left(\frac{-26830}{T_s} + 30.114\right) [\text{bar}^2]$
2	$K_{II} = K_I \cdot K_{III} [\text{bar}^2]$
3	$K_{III} = \exp\left(\frac{4400}{T_s} - 4.036\right) [\text{bar}^2]$

Table 1: Reaction equilibrium constants

The determined reaction rates are used for calculating the rate of formation of each substance. This is done according to [7]:

$$r_{CH_4} = -v_1R_1 - v_2R_2 - v_4R_4 \quad (34)$$

$$r_{O_2} = -2v_4R_4 \quad (35)$$

$$r_{CO_2} = v_2R_2 + v_3R_3 + v_4R_4 \quad (36)$$

$$r_{H_2O} = -v_1R_1 - 2v_2R_2 - v_3R_3 - 2v_4R_4 \quad (37)$$

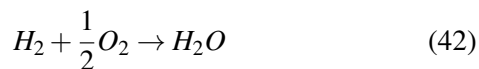
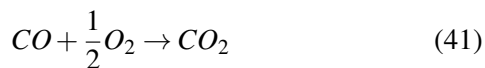
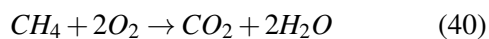
$$r_{H_2} = 3v_1R_1 + 4v_2R_2 + v_3R_3 \quad (38)$$

$$r_{CO} = v_1R_1 - v_3R_3 \quad (39)$$

The v factors are effectiveness factors used to account for the intraparticle transport limitation. Their values are set to $v_1 = 0.07$, $v_2 = 0.06$, $v_3 = 0.7$ and $v_4 = 0.05$, which are average values obtained from simulations of the various species inside the Ni catalyst pellet [8]. The low values for all reactions except the water gas shift reflect the severe limitation on the catalytic conversion of methane to synthesis gas due to intraparticle diffusion in the catalyst pellets. Calculation of effectiveness factors using various methods and a comparison of their effect on simulation results is presented in [9].

5.2 Catalytic burner

The catalytic burner model includes reactions for oxidation of methane (40), carbon monoxide (41) and hydrogen (42) [4]:



The implementation of the burner is based on a volume model with air and fuel inlet ports and an air outlet port. It also has a port for heat flow to the environment and a boolean input for ignition signals. The burner operates in one of two states, either burning or mixing, where the difference is the stoichiometry matrix defining how the fuel components is mixed into the air flow. The burner can be ignited if the air/fuel ratio is low enough and the ignition signal is true. The burner is set to mixing state if the air/fuel ratio gets too high.

When combustion occurs a combustion rate defined as the fraction of fuel burned is calculated dynamically.

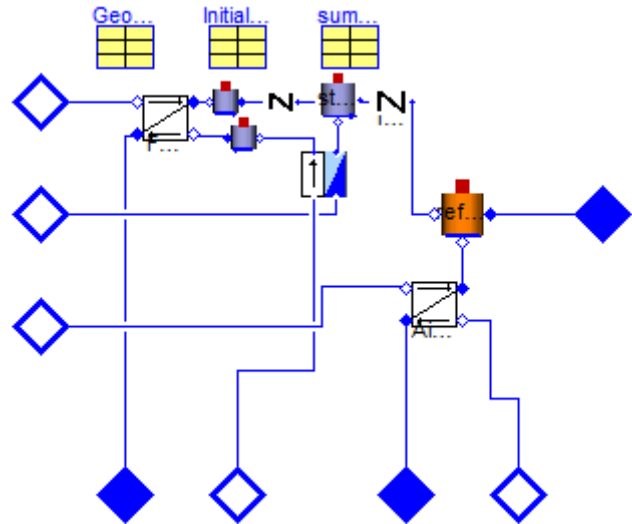


Figure 5: Graphical layout of the hotbox

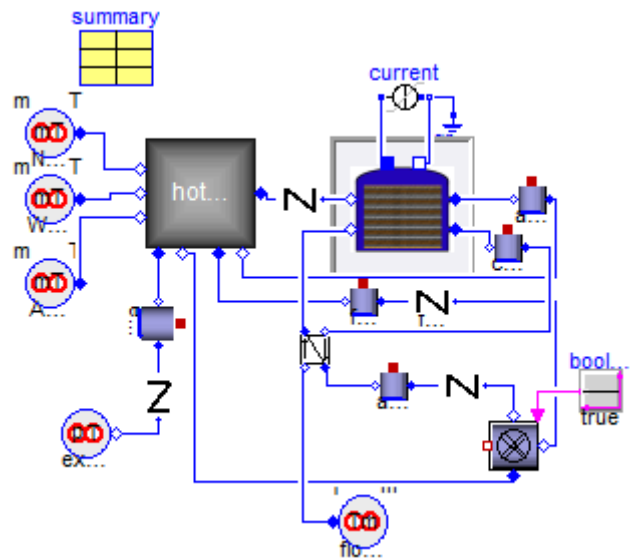


Figure 6: Graphical layout of the system model

5.3 System implementation

The system model is implemented in Dymola. Figure 5 shows a hotbox model, which includes the ATR unit, the steam generator, a mixing volume for natural gas and steam, and two heat exchangers for heat recirculation. In Figure 6 the complete system model is shown. It includes the hotbox and stack models, as well as the catalytic burner, a heat exchanger and flow sources of natural gas, water and air. Note that the implemented system uses co-flow of fuel and air in the stack, counter-flow can be obtained by reversing the air flow direction.

Parameter	Value
Anode thickness	1 mm
Cathode thickness	70 μm
Electrolyte thickness	10 μm
Cell pitch length	5 mm
Active cell area	361 cm^2
Fuel molar fraction	26.3% H_2 , 17.1% CH_4 , 2.9% CO , 4.4% CO_2 , 49.3% H_2O
Air oxygen fraction	23%

Table 2: Cell geometry and fuel composition used in simulation

6 Results

Several simulations of the complete system and individual components are carried out in order to show their behaviour. Three of them are presented here. In all simulations the cells were discretized with four elements along the fuel flow direction. The effects of varying the number of elements has been investigated and four elements were found to give a good compromise between accuracy and computation time. When changing from 4 to 50 elements the resulting difference in cell voltage was 1.4%, and the difference in total heat production was 1.8%. If the purpose of the simulation is to locate hotspots or other local phenomena in the cell, more elements should be used. The cell geometry parameters and fuel composition used during the simulations are presented in Table 2. The values correspond to a 5 kW SOFC stack manufactured by Forschungszentrum Jülich, Germany, which is described in [2, 4].

6.1 Substack with simplified cell model

In this simulation the substack with the simplified cell model is simulated with a constant inlet flow rate of fuel and air. During the simulation the external current is increased linearly from 1 to 150 A, which corresponds to a cell current density of 27.7 to 4155 A/m^2 . The resulting cell voltage and power generation is shown together with simulation results from [4] in Figure 7. The reference model is a 0D SOFC model where the Nernst potential is evaluated using an average fuel composition between the stack inlet and outlet. The marked upper and lower limits were obtained using the fuel composition at the stack inlet and outlet respectively. The difference between the results are not very large, but except for the different modeling approaches also different handling of heat contributes to

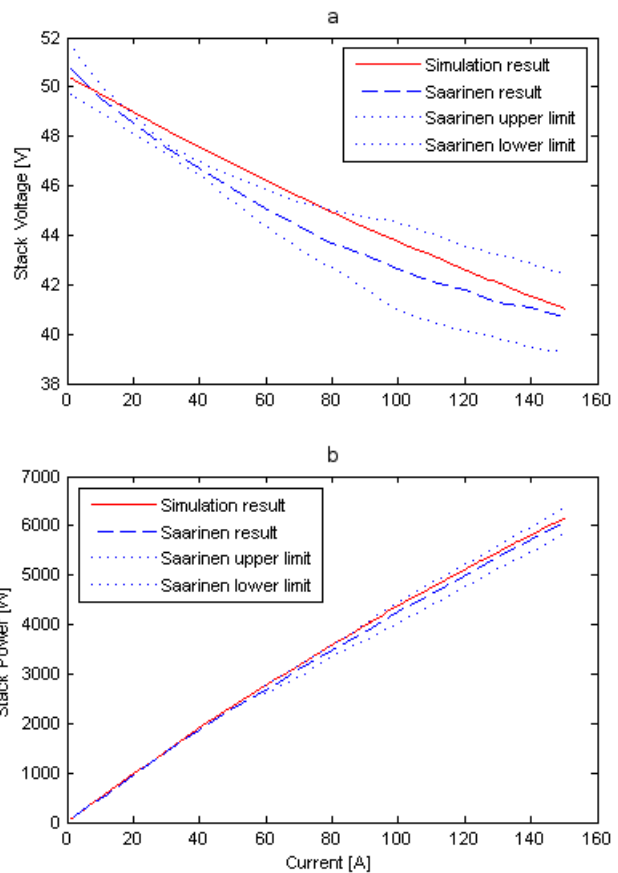


Figure 7: Simulation results for: (a) cell voltage, (b) cell power generation. Results from [4] is included for comparison.

the deviation. In the reference model the temperature was kept constant at 760°C , whereas in our model the stack was cooled only by heat transfer between the gases and the inner surfaces of the fuel and air channels. This resulted in an increasing stack temperature with increased current, varying from 750°C to 810°C .

6.2 Substack with complete cell model

In this simulation the substack with the complete cell model is simulated with a constant inlet flow rate of fuel and air. During the simulation the external current is increased linearly from 10 to 520 A, which corresponds to a cell current density of 277 to 14404 A/m^2 and fuel utilization of 30% to 100% for this specific cell.

In Figure 8(a) the cell voltage versus the cell current density shows the expected characteristics with a fast voltage drop for low current densities, a close to linear behaviour for medium current densities and a faster drop at high current densities. In Figure 8(b) the cell

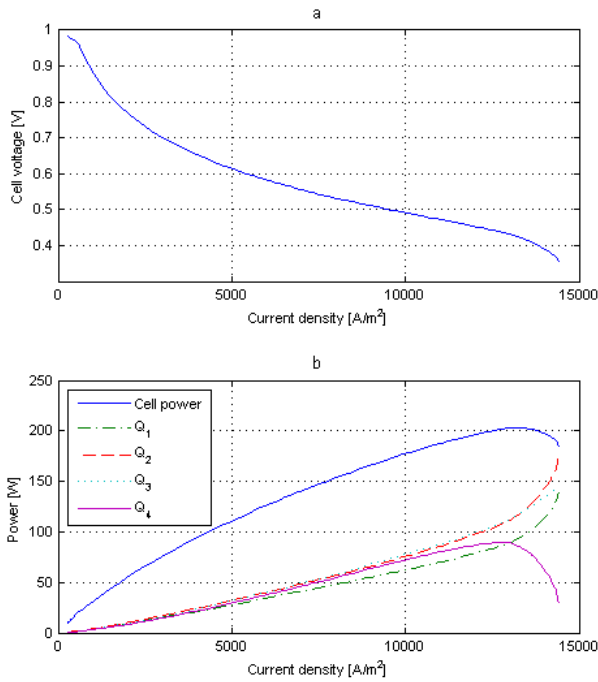


Figure 8: Simulation results for: (a) cell voltage, (b) cell power production and heat production in the four discrete elements

shows maximum power production at 13000 A/m². Also shown is the heat production in every discrete element, numbered from inlet to outlet. The heat production grows with increased current density and is roughly the same in each discrete element, except for very high currents when the heat production in the 4th element decreases rapidly. Figures 9(a)-(d) shows the fuel and air composition in every discrete element, numbered from inlet to outlet. The gradual consumption of hydrogen, methane and oxygen, and production of steam, can be observed.

6.3 Complete system transient

The system in Figure 6 is simulated with a constant external current of 150 A, corresponding to a cell current density of 4155 A/m². Three substacks are included, the middle one modeling 30 cells, and the top and bottom ones modeling 10 cells each. The inlet flow rates to the hotbox are chosen such that a steam carbon ratio of 1.2 and a oxygen gas carbon ratio of 0.2 is achieved in the autothermal reformer, and the stack temperature is initialized at 800°C. The simulation results for the system transient behaviour are presented in Figures 10 - 14. The substacks are numbered in the fuel flow direction of the manifold volume, i.e. substack 1 has the highest inlet pressure. In Figure 10 the middle substack

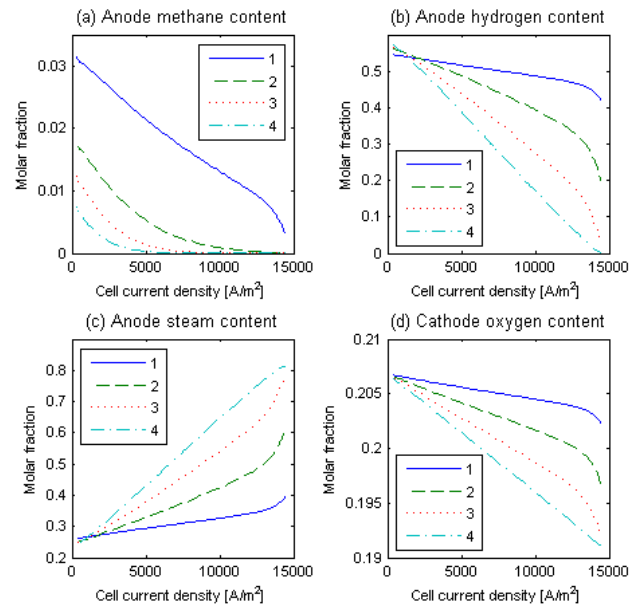


Figure 9: Simulation results for: (a) anode methane content, (b) anode hydrogen content, (c) anode steam content, (d) cathode oxygen content. Results shown for each discrete element, numbered from inlet to outlet.

shows the highest voltage per cell, due to the higher temperature than in the other substacks, as shown in Figure 11(a). The voltage losses in the cell decrease with increased temperature due to higher conductivity of the cell materials, and increased exchange current density which gives lower activation losses. The fuel utilization, shown in Figure 11(b), is highest in the lowermost substack, due to the pressure drop in the manifold volumes, which gives lower mass flow rates in the subsequent substacks. Presented in Figure 12 are the power and heat generation of the stack.

The resulting temperature profiles for the ATR unit and the catalytic burner are presented in Figure 13, together with the conversion ratios of methane and oxygen in the ATR. The oxygen conversion ratio is 1 except for the the first seconds after start up. This is desirable, as oxygen in the stack inlet fuel flow decreases the overall system efficiency and destroys the stack as the Ni catalyst is oxidized. The high temperature of the burner is required for steam generation and pre-heating in the hotbox. The conversion ratio of methane in the ATR gives the stack inlet fuel composition presented in Figure 14. The composition reaches its steady-state value quickly, and then varies slightly with temperature. This composition complies with the results from similar simulations in [4].

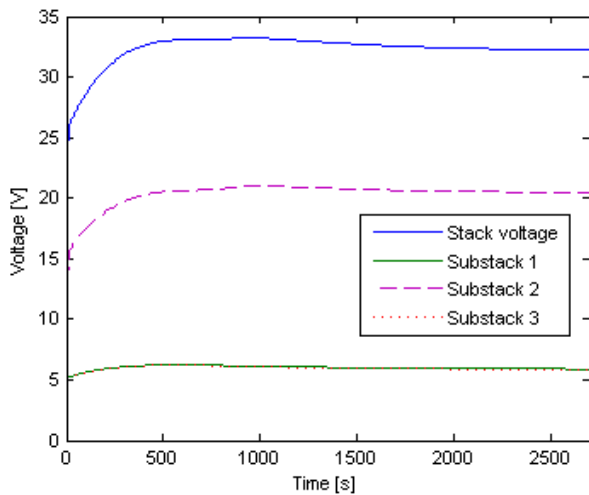


Figure 10: Dynamic simulation results for the voltage of the stack and individual substacks

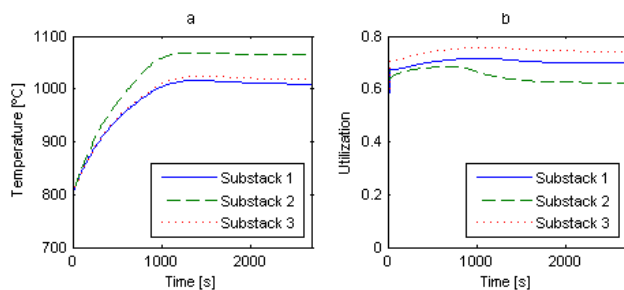


Figure 11: Dynamic simulation results for: (a) temperatures of the substacks, (b) hydrogen utilization of each substack

7 Conclusions

It is concluded that Dymola and Modelica are very useful tools for fuel cell system modeling. The positive aspects are mainly the equation based language, the multi domain possibilities, and the flexibility of an object oriented structure.

All developed components were simulated individually for verification. The comparison of the substack with the simplified cell model is presented in this paper. For the complete system it has not been possible to make relevant comparisons of absolute values, as geometry parameters and correlations for different components comes from different sources. The objective of this study was to assure that results obtained are phenomenologically sound. This was verified by running several simulations of the developed components under various boundary conditions and studying the influence on the results. The results of these simulations are presented in the original thesis [10].

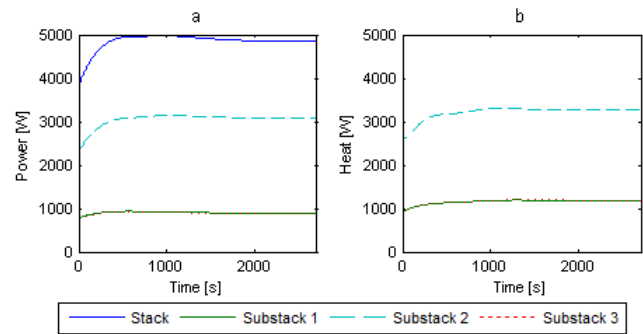


Figure 12: Dynamic simulation results for: (a) power generation of stack and individual substacks, (b) heat production of the substacks

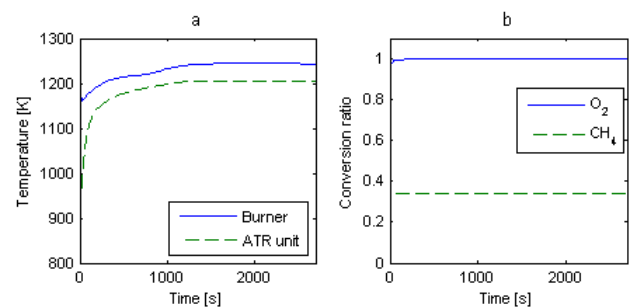


Figure 13: Dynamic simulation results for: (a) burner and ATR unit temperature, (b) ATR conversion ratio of methane and oxygen

As discussed in the abstract and introduction, modeling all relevant physics in a fuel cell system requires a flexible tool capable of including several physical domains in the same model. The equation based language and possibility for acausal models allows for physical models, which is very useful if components are to be reused in multiple system designs with different purposes. The system model developed in this study includes the electrochemistry of the cell, dynamic chemical reactions in the ATR unit as well as in the cell flow channel and the catalytic burner. It also includes heat exchanger models and a small electrical system. The system could easily be expanded to include a physical model of an electric load, or redesigned to another system configuration using the same component models.

One negative aspect can be mentioned. There is no automatic way to perform discretization of partial differential equations in Dymola, as some other non-Modelica based tools provide. The discrete equations must be stated directly by the user which makes it harder and more time consuming than using tools de-

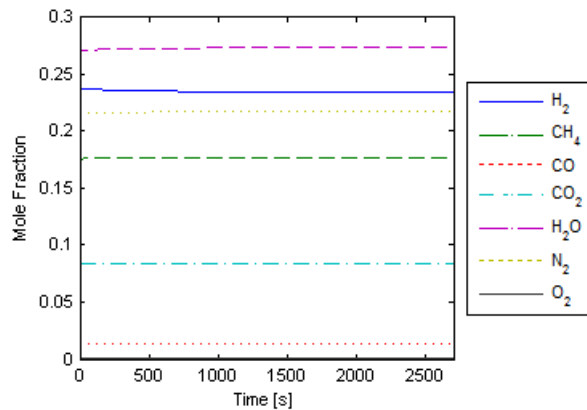


Figure 14: Dynamic simulation results for the ATR outlet fuel composition

veloped for this purpose. On the other hand this approach does not impose any limitations on how discretizations can be made and is thereby useable for many different applications.

References

- [1] J. Larminie and A. Dicks. *Fuel Cell Systems Explained, Second Edition*. Wiley, 2003. ISBN 0-470-84857-X.
- [2] M. Kemm. *Dynamic Solid Oxide Fuel Cell Modelling for Non-steady State Simulation of System Applications*. PhD thesis, Division of Thermal Power Engineering, Lund University, 2006.
- [3] S.H. Chan, H.K. Ho, and Y. Tian. Modelling of simple hybrid solid oxide fuel cell and gas turbine power plant. *Journal of Power Sources*, 109:111–120, 2002.
- [4] J. Saarinen, M. Halinen, J. Ylijoki, M. Noponen, P. Simell, and J. Kiviaho. Dynamic model of 5 kW SOFC CHP test station. *Journal of Fuel Cell Science and Technology*, 4:397–405, 2007.
- [5] P. Costamagna, A. Selimovic, M. Del Borghi, and G. Agnew. Electrochemical model of the integrated planar solid oxide fuel cell (IP-SOFC). *Chemical Engineering Journal*, 102:61–69, 2004.
- [6] E. Fontell, T. Kivisaari, N. Christiansen, J.-B. Hansen, and J. Pålsson. Conceptual study of a 250 kW planar SOFC system for CHP application. *Journal of Power Sources*, 131:49–56, 2004.
- [7] M.H. Halabi, M.H.J.M. de Croon, J. van der Schaaf, P.D. Cobden, and J.C. Schouten. Modeling and analysis of autothermal reforming of methane to hydrogen in a fixed bed reformer. *Chemical Engineering Journal*, 137:568–578, 2008.
- [8] Ann M. De Groote and Gilbert F. Froment. Simulation of the catalytic partial oxidation of methane to synthesis gas. *Applied Catalysis A: General*, 138:245–264, 1996.
- [9] Krzysztof Gosiewski, Ulrich Bartmann, Marek Moszczyński, and Lesław Mleczko. Effect of the intraparticle mass transport limitations on temperature profiles and catalytic performance of the reverse-flow reactor for the partial oxidation of methane to synthesis gas. *Chemical Engineering Science*, 54:4589–4602, 1999.
- [10] Daniel Andersson and Erik Åberg. Dynamic modeling of a solid oxide fuel cell system in Modelica. Master's thesis, Department of Energy Sciences, Lund University, 2010.

The OnWind Modelica Library for Offshore Wind Turbines - Implementation and first results

M. Strobel F. Vorpahl C. Hillmann X. Gu A. Zuga U. Wihlfahrt
Fraunhofer Institute for Wind Energy and Energy System Technology (IWES)
Michael.Strobel@iwes.fraunhofer.de

Abstract

At Fraunhofer IWES a Modelica Library including all major components needed for load calculations of current offshore wind turbines is developed. The library additionally includes models for external conditions, like wind, soil and waves, and their respective influence on the structures. The library constitutes a large effort in the creation of a highly coupled multi-physics model with Modelica for an industrial project. The results obtained with this library are compared to the results from the IEA Wind Task 23 project OC3¹ (Offshore code comparison collaboration). The OC3 project is an international effort to define a set of load-cases and a reference wind turbine that are used to verify simulation systems on a code-to-code basis. In this paper the status and the implemented theories of the individual models at IWES are explained and verification results are presented and discussed.

Keywords: offshore wind turbine simulation; aerodynamics; hydrodynamics; OC3 project; fully coupled simulation

1 Introduction

The use of wind energy as a renewable and comparably cost effective energy source has grown rapidly over the past decades. While today the majority of the turbines are mounted onshore, we are now facing a rapid development offshore as well.

This paper starts with an introduction to wind turbine simulation in general before describing the components of the OnWind library. In the last section the simulation results of this library are verified in comparison to results obtained in the OC3 project.

2 Wind turbine system simulation

Wind turbines are designed and analyzed using simulation tools (i.e. design codes) capable of predicting the coupled dynamic loads and responses of the system. Land-based wind turbine analysis relies on the use of aero-servo-elastic codes, which incorporate wind-inflow, aerodynamic (aero), control system (servo) and structural-dynamic (elastic) models in the time domain in a coupled simulation environment. In recent years, some of these codes have been expanded to include the additional dynamics pertinent to offshore installations, including the incident waves, sea current, hydrodynamics and foundation dynamics of the support structure.

The design of wind turbines usually consists of a two step approach, starting with a fully coupled time domain simulation as - described above - which generates the loads for the following detailed design of all components. The certification rules for offshore wind turbines, like the “Design requirements for offshore wind turbines” issued by the International Electrotechnical Commission [7], define a large set of load cases that have to be considered in the design phase.

3 Components of the library

Due to the object-oriented structure of Modelica, the different components needed for the simulation of the turbine can be implemented independently. In this chapter the different components are described in brief.

The structure of the OnWind library is shown in Figure 2.

3.1 Wind

In the OnWind library, wind models for simple cases including exponential wind shear are available. For

¹www.ieawind.org/

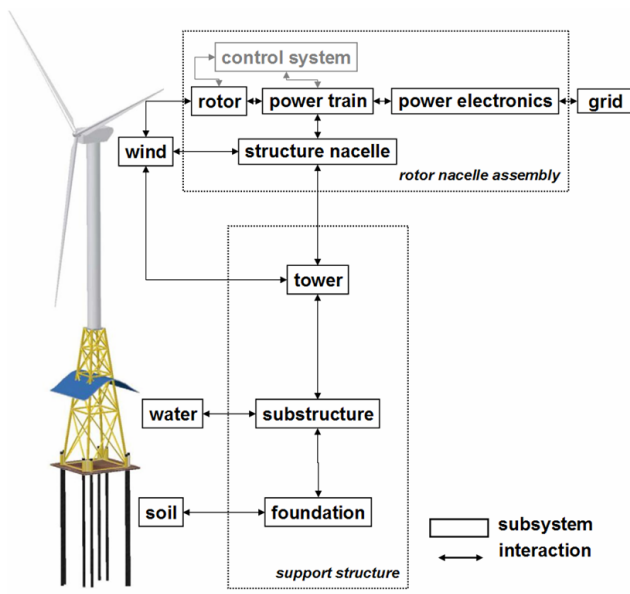


Figure 1: Major components of an offshore wind turbine

certification purposes even more complex wind models are needed which are currently under development. As these stochastic models for turbulent wind definition (Kaimal [11] and Von Karman spectra [12]) are available as open source free-to-use FORTRAN software from the National Renewable Energy Laboratory of the US (TurbSim, [8]), this software is currently used for the calculations. The interface between TurbSim and the wind turbine model is part of the OnWind library. For the integration of these external wind files an optimal tradeoff between memory consumption and calculation performance is achieved.

3.2 Aerodynamics

A basic standard for wind turbine simulation tools is the “Blade element momentum theory (BEM)”, as described by [17], [4] and extended by [3], [6], which is used to calculate induced wind velocities and aerodynamic loads. This method is incorporated in the library including the blade tip losses, the hub losses [6] and corrections for the turbulence wake state [5]. Additionally the tower interference is accounted for in the aerodynamic model.

3.3 Rotor blade structure

For the modeling of the structure of the rotor blades there are currently two models available, a rigid model and a flexible one. The rigid model can be used for rough calculations and has significant advantages in

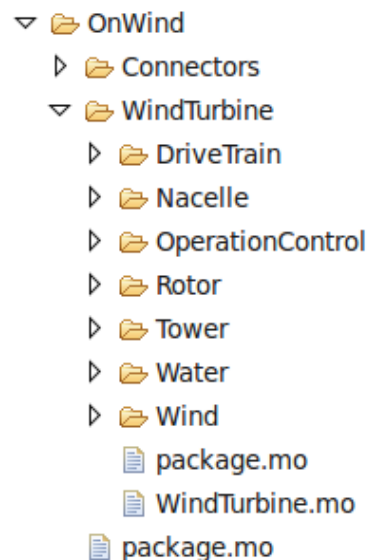


Figure 2: Structure of the OnWind library

terms of performance. For the flexible blades a finite-element formulation based on Euler-Bernoulli beam theory is implemented. The structural analysis and the aerodynamic calculations are coupled in this model to simulate the flexible rotor.

3.4 Hub and nacelle

As in most software systems for fully-coupled wind turbine simulation the hub and the nacelle are modeled as rigid links contributing masses and inertias.

3.5 Drive train and generator

The drive train is incorporated in the library as a two mass torsional oscillator, contributing inertia, gearbox ratio, stiffness and damping. For more accurate and computational costly simulations of the drive train the DLR drive train library can be used, as it is compatible to the OnWind library.

As in most of the wind turbine simulation codes the generator is included with its mechanical characteristics, but more advanced models from the Modelica Standard Library or other libraries may be used.

3.6 Control

The control module of the OnWind library consist on the one hand of control sequences to handle different operating conditions and on the other hand of conventional methods to control different parts of the wind turbine.

In the operation control package relevant inputs and outputs are processed to adjust corresponding operating states of the wind turbine. This is done by a simple decision algorithm but can also be realized using state charts with e.g. the MODELICA StateChart2 library [15]. To retrieve measured process signals and transferring controller output signals, the communication with sensors and actuators is accomplished by the signal bus concept of MODELICA as described by Martin Otter [14]. This eases the integration of already implemented or new signals into and out of the control modules. Before processing input signals in the control algorithms preprocessing functions like e.g. low pass filtering of generator speed is performed.

Control methods are implemented for blade pitching and controlling the generator torque. For aerodynamic power control a collective pitch control with PI-algorithm is used. To account for the non-linear behaviour of aerodynamic profiles a gain-scheduling algorithm is used like discussed in [9]. Limit conditions are also considered and other linear control structures like PID and Anti-Wind-Up may be used.

To accomplish generator variable-speed operation and energy capture maximization it is possible to set a characteristic generator torque vs. speed curve. The curve is the basis for setting the generator torque on different speeds by a suitable algorithm. For performance optimization also a Maximum-Power-Point-Tracking algorithm is implemented, which calculates the optimal speed for changing operating conditions.

3.7 Tower and substructure

For the tower and the substructure a finite element approach is used (Euler-Bernoulli beam elements). This allows for modeling of arbitrary support structures, like the jacket shown in Figure 3. Simple monopiles can be modeled with this approach as well.

For the calculation of the forces resulting from water waves and currents, Morison's formula [13] is incorporated in the substructure models.

3.8 Soil and water

Several currently available tools for wind turbine simulation model the soil rather simple with one 6x6 stiffness matrix or just as a clamped beam. In OnWind this is possible too, but the more advanced p-y-approach is incorporated as well. With this approach, also described by the American Petroleum Institute ([2]), the soil is modeled with nonlinear springs. Additionally



Figure 3: Turbine on jacket substructure extended by tubular tower (prototype erected onshore)

damping elements are available to describe the soil as realistic as possible.

Using the OnWind library single linear waves (Airy [1]) from different angles may be defined. Non-linear waves based on a Stream Function theory are available as well as spectrum based linear irregular waves defined by means of JONSWAP spectra. Development and first verification of hydrodynamic features are described in [16].

3.9 Connectors

Most of the wind turbine components described earlier in this document use predefined connectors like the frame connector from the multibody library to connect adjacent components of the wind turbine. Therefore it is easily possible to integrate enhanced models for subcomponents using parts of the Modelica Standard Library. However some of the wind turbine components need special connectors to be able to model the interaction between each other.

Especially the connections between the components modeled with the finite-element approach (rotorblade structure, tower and substructure models) and the models describing the loads on these components (wind and wave models) needed to be developed.

For example the connection between the wind and the rotorblade is realized by two connectors connecting the wind model via the aerodynamics model to the rotorblade model like shown in Figure 4. For the con-

nection between the wind model and the aerodynamics model a connector has been developed containing the actual position of the structure and the velocities of the wind. The forces calculated in the aerodynamics model are connected to the structure model via another connector containing the position of the structure and the forces applied on the structure. The same approach is used to describe the interaction between the water model and the models for the tower and the sub-structure.

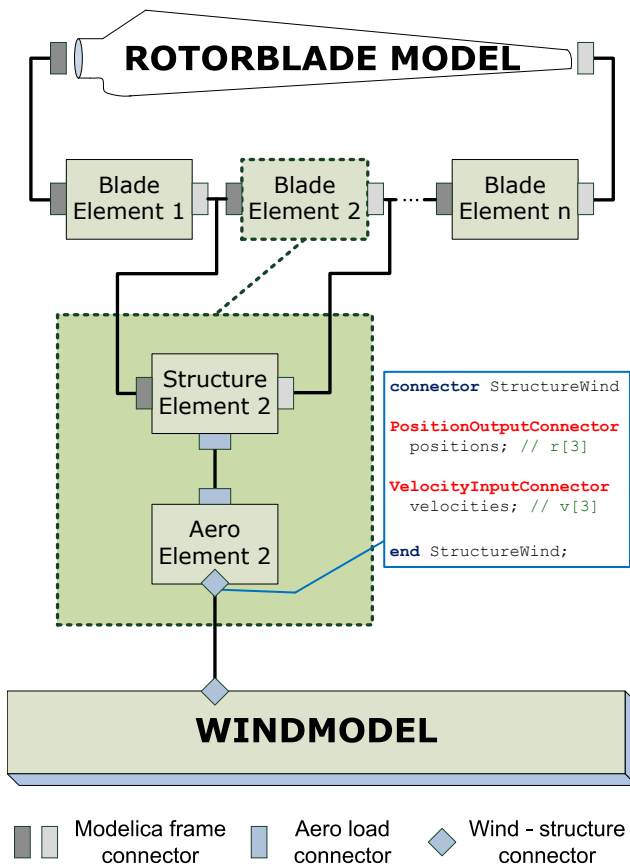


Figure 4: Connector schema for rotorblade and wind

4 Verification of the OnWind library

In this chapter, the offshore wind turbine that is used in this benchmarking task and the OC3 project - in which e.g load cases and support structures are defined - is briefly described. Furthermore, results calculated with OnWind are presented and interpreted in comparison to results from other tools.

4.1 The NREL 5-MW baseline wind turbine

The NREL 5-MW baseline wind turbine developed for code comparison purposes is a lifelike three-bladed variable speed 5-MW upwind turbine with collective pitch control. The model is based on available design information from turbine manufacturers with an emphasis on the REpower 5M machine. Detailed data is provided by research projects - with a focus on the Dutch Offshore Wind Energy Converter (DOWEC) project - where design data is not available due to confidentiality reasons. Special emphasis was set on combining the best available and the most representative data. The turbine definition includes aerodynamic and structural data as well as the definition of a basic control-system. For more details on the turbine cf. [9].

4.2 The OC3 project

The sophistication of the aero-hydro-servo-elastic codes (as described in section 2), and the limited data available with which to validate them, provided the incentive to set up an international code-to-code comparison project in order to validate these tools. This task was accomplished in the Offshore Code Comparison Collaboration (OC3) project, which operated under Subtask 2 of the International Energy Agency (IEA) Wind Task 23 [10].

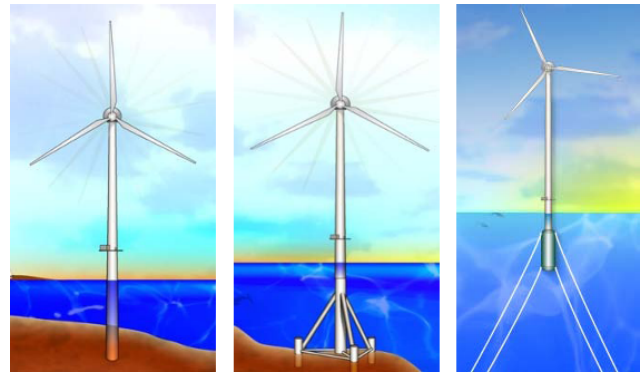


Figure 5: The support structures investigated in the OC3 project [10]

The general approach of the project is to (1) discuss modeling strategies, (2) develop a suite of benchmark models and simulations, (3) run the simulations and process the simulation results and (4) compare and discuss the results.

A large set of time series simulation results such as turbine operational characteristics, external conditions as well as load and displacement outputs were compared and interpreted. Load cases were defined and

run with increasing complexity to be able to trace back differences in simulation results to the underlying error sources.

Within the project, the NREL 5-MW baseline wind turbine is modeled in combination with four different substructure types in four phases: The monopile implemented in Phase I is designed for a water depth of 20m with its transition to the tower 10m above the mean sea level (MSL). This structure is used for the validation in the following chapter.

4.3 Results comparison

The loadcases in Phase I of the OC3 project are designed in different stages, from rigid structure over a flexible onshore model to a flexible offshore turbine with additional hydrodynamic loads. In the following, exemplary time series results from the last stage of a fully coupled flexible offshore model with deterministic wind and wave conditions are presented. For the following comparison the results of three well known participants (GL Garrad Hassan, NREL, Risø DTU) of the OC3 project are taken into account ².

In Figure 6 and 7 the gear-box translated generator speed and the generated power of the turbine at a constant wind speed of 8 m/s is shown.

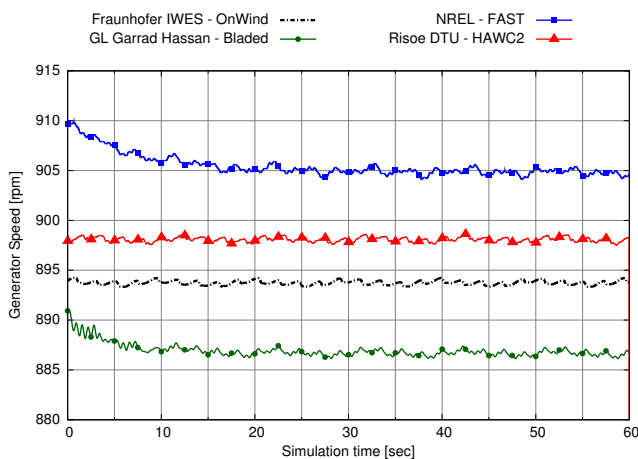


Figure 6: Time series of generator speed in [rpm]

The results of the OnWind library are located in a range of 2% difference for the absolute values which is due to different implementations of the aerodynamics model. The main reason for the oscillating behaviour is the influence of the tower shadow.

²http://www.ieawind.org/Annex_XXIII.html

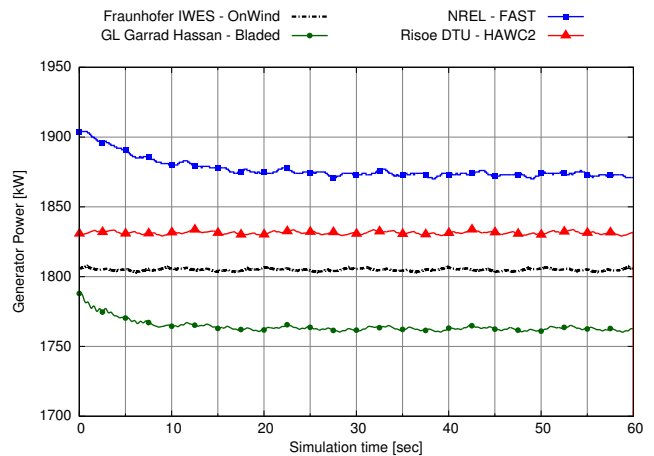


Figure 7: Time series of electrical generator power output in [kW]

As we deal with a flexible structure, deflections of the blade tip are illustrated in the next diagrams. Both results, the out-of-plane (Figure 8) and the inplane (Figure 9) deflections are in good agreement. The smoother form of the deflection curves from OnWind is the result of not yet included extensions to the aerodynamical model like dynamic stall and correction for skewed inflow, which are currently in development.

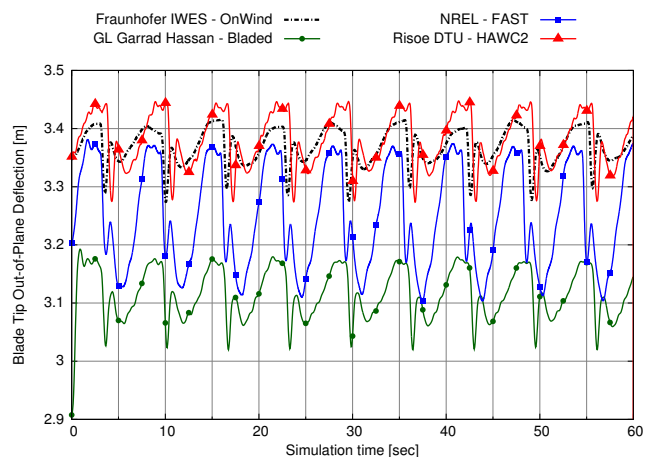


Figure 8: Time series of out-of-plane blade tip deflection in [m]

As mentioned above, the investigated loadcase encloses hydrodynamic loads from deterministic wave conditions with a regular wave of 6m elevation and a period time of 10 seconds. Figure 10 shows the base shear force on the monopile mudline location to demonstrate the effects of wave loads.

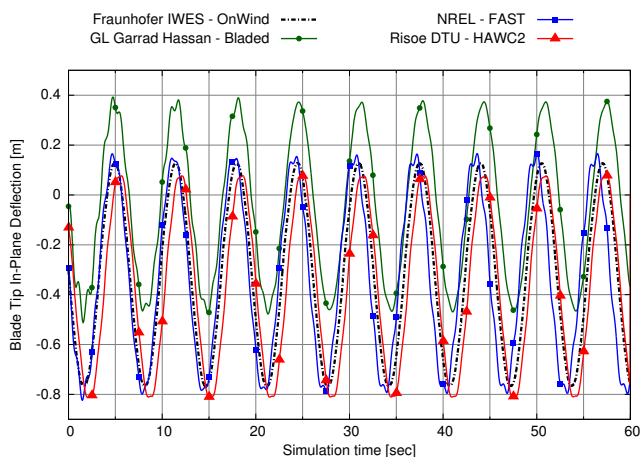


Figure 9: Time series of in-plane blade tip deflection in [m]

In summary it can be said, that the results of the On-Wind library compare well to the results of Phase I of the OC3 project. As the development of the OnWind library is still in progress, the small deviations that occurred with respect to other simulation tools are subject to ongoing research.

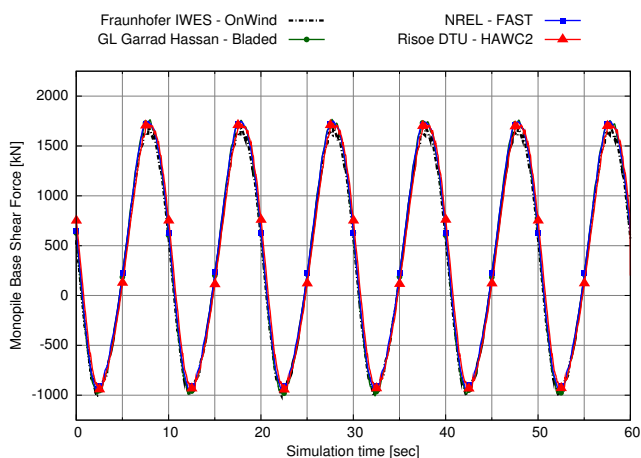


Figure 10: Time series of monopile base shear force [kN] in wind direction

5 Conclusion

It is shown that the Modelica Language is a good choice to develop a library for offshore windturbines. Especially the hydro-aero-servo-elastic coupling of the different components of a wind turbine is easy to implement. With respect to the short development time very good results compared to other established tools on the market are achieved.

The ongoing development of the OnWind library at Fraunhofer IWES concentrates on implementation of further substructure designs like tripod, jacket and

floating structures in order to complete the comparison with the OC3 project. For these advanced structures the Timoshenko beam theory needs to be implemented. Additionally the integration of structural components as mode shapes is planned.

Furthermore investigations on further enhancements of the aerodynamic model like the dynamic stall and the wake model are in progress.

References

- [1] G.B. Airy. On tides and waves. In *Encyclopaedia Metropolitana*, 1845.
- [2] American Petroleum Institute (API), Washington DC, USA. *RP 2A-LRFD: Recommended Practice for Planning, Designing and Constructing Fixed Offshore Platforms - Load and Resistance Factor Design*, 1993.
- [3] A. Betz. Das Maximum der theoretisch möglichen Ausnützung des Windes durch Windmotoren. *Zeitschrift für das gesamte Turbinewesen*, 26:307–309, 1920.
- [4] R.E. Froude. On the part played in propulsion by differences of fluid pressure. *Transactions of the Institution of Naval Architects*, 30:390–405, 1889.
- [5] H. Glauert. A general theory of the autogyro. *ARCR R&M*, 1111, 1926.
- [6] H. Glauert and L. Division. *Airplane Propellers, Aerodynamic Theory*, volume 4. Durand WF, Berlin, 1935.
- [7] IEC. *Wind turbines - Part 3: Design requirements for offshore wind turbines*. IEC 61400-3, 1.0 edition, 2009.
- [8] J. Jonkman. *TurbSim User's Guide Version 1.5*. National Renewable Energy Laboratory (NREL), Golden, Colorado, USA, 2009.
- [9] J. Jonkman, S. Butterfield, W. Musial, and G. Scott. Definition of a 5-MW reference wind turbine for offshore system development. Technical report, National Renewable Energy Laboratory (NREL), Golden, Colorado, USA, 2009.
- [10] J. Jonkman and W. Musial. IEA wind task 23 subtask 2: The offshore code comparison collaboration (OC3). Technical report, National

Renewable Energy Laboratory (NREL), Golden, Colorado, USA, 2010.

- [11] J.C. Kaimal, J.C. Wyngaard, Y. Izumi, and O.R. Coté. Spectral characteristics of surface-layer turbulence. *Quarterly Journal of the Royal Meteorological Society*, 98(417):563–598, 1972.
- [12] T. von Kármán. Progress in the statistical theory of turbulence. In *Proceedings of the National Academy of Sciences of the United States of America*, volume 34, pages 530–539, August 1948.
- [13] J.R. Morison, M.P. O'Brien, J.W. Johnson, and S.A. Schaaf. The force exerted by surface waves on piles. *Petroleum Transactions, AIME*, 189:149–154, 1950.
- [14] M. Otter. Modeling, Simulation and Control with Modelica 3.0 and Dymola 7. Technical report, Deutsches Zentrum fuer Luft- und Raumfahrt e.V. DLR - Institut fuer Robotik und Mechatronik, Wessling, Germany, 2009.
- [15] M. Otter, M. Malmheden, H. Elmquist, S.E. Mattsson, and C. Johnsson. New formalism for modeling of reactive and hybrid systems. In *Proceedings of the 7th Modelica'2009 Conference*, Como, Italy, 2009. The Modelica Association.
- [16] L. Quesnel, F. Vorpahl, and M. Strobel. Hydrodynamics meet wind turbines: specification and development of a simulation tool for floating wind turbines with modelica. In *Proceedings of the 20th International Offshore and Polar Engineering Conference*. Fraunhofer Institute for Wind Energy and Energy Systems Technology (IWES), 2010.
- [17] W.J. Rankine. On the mechanical principles of the action of propellers. *Transactions of the Institution of Naval Architects*, 6:13–30, 1865.

Modeling of Gas-Particle-Flow and Heat Radiation in Steam Power Plants

Leo Gall Kilian Link Haiko Steuer

Siemens AG, Energy Sector, Erlangen, Germany

leo.gall@mytum.de, kilian.link@siemens.com, haiko.steuer@siemens.com

Abstract

Fired steam generators are the dominating technology of coal combustion in power plants. This paper presents a model for pulverized coal fired steam generators in Modelica. The model components are designed as an extension of SiemensPower, a Modelica library for transient simulation of power plants.

The focus is on coal combustion, gas-particle-flow and radiation heat transfer in the furnace. The dispersed flow of flue gas has to be modeled because radiation heat transfer gets considerably intensified by the contained coal and ash particles. Customized connectors for the dispersed flow of flue gas had to be developed on the base of Modelica.Fluid.

The component oriented approach supports the adaptation of the model to different simulation tasks, such as stability analysis of the evaporator or influence of the coal mill on plant dynamics. Component structure, parametrization and spatial discretization were important aspects for the development of maintainable and re-usable components of the model library.

Keywords: steam power plant; coal; dispersed flow, radiation heat transfer; connectors

1 Introduction

Coal fired steam generators are widely used in commercial power plants around the world. Even in 20 years, more than one third of the world's growing electric energy demand will be produced out of coal [6]. Today's coal power plant units generate up to 1100MW of electric power and steam generators reach heights of 170m.

Before combustion, the coal gets pulverized in coal mills to particle sizes of about $50\mu\text{m}$. Some of the combustion air is used to transport the coal dust into

the furnace. In the furnace, the coal is burned while moving within the flow of combustion air and flue gas. The combustion takes about one second in most cases, but the burn-out of larger particles can take longer.

Furnaces for hard coal with dry ash removal were modeled first, as they are the dominating technology today. The model could be adapted to lignite plants by adding equations for flue gas recirculation. If necessary, slag-tap furnaces can be modeled, too.

Flexibility is the main challenge in development of coal power plants. For fired steam generators this implies more dynamic operation and usage of different coal types and biomass.

A detailed model of the steam generator is needed, in order to analyze the dynamic behavior. The heat release of the combustion is non-uniform over the height of the furnace. This non-uniform heat flow into the water wall tubes influences the flow stability and the energy storage of metal and fluid. The coal mills define the speed of fuel ramps and the distribution of coal particle sizes. The furnace model allows to study the impact of coal mill dynamics on power output.

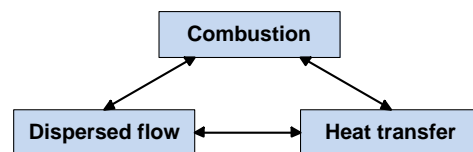


Figure 1: Dependencies of modeling domains

The focus of the presented work is on modeling of the furnace and radiation heat transfer inside of the furnace [4]. Extensive simplification has to be applied in order to achieve a model for dynamic simulation tasks on system-level. Power plant furnaces are large systems with turbulent reactive flow in three dimensions. Spatial discretization was applied in order to reduce the partial differential equations into a set of ODEs which can be solved by a Modelica tool. The guide-

line for model assumptions were proven steady-state inhouse tools. A trade-off between model accuracy and numerical speed and robustness has to be made, if the furnace model gets coupled to the water/steam-cycle.

For practical simulation tasks, the available parameter set and measurement data for calibration is very limited.

Figure 1 shows the needed models of coupled physical phenomena for the furnace model. The combustion model calculates the heat release (heat flow and burn out) and flue gas composition. This depends directly on the model of dispersed flow which fullfills the mass, energy and momentum balances. The resulting radiation heat transfer depends on the temperatures of flame and wall. Especially in transparent flame zones, flue gas composition and particle properties influence emission and absorption coefficients.

2 Modeling Framework

SiemensPower is a growing specialized model library for transient simulation of power plants in Modelica. The models are used to enhance the performance in operation and control of existing and new power plants. The components of the library are designed for many different power plant applications. One example are combined cycle gas turbine (CCGT) plants with heat recovery steam generators. The library is used as a basis for studies of multiple engineering design questions. SiemensPower is based on Modelica.Fluid connectors and Modelica.Media. Dymola [5] is used as the main tool for modeling and simulation. The model components of fired steam generators add a new field of application to the SiemensPower library.

Besides Modelica/Dymola there are several inhouse simulation tools in use at Siemens Energy which have to be maintained. If the Modelica model of the fired steam generator delivers reliable results, older inhouse tools could be potentially replaced. Other specialized inhouse tools could be coupled to the Modelica model in order to extend their application range.

The existing steady state tools were used as a guideline for component structure design, parametrization and validation of the new Modelica model.

3 Model Description

3.1 Model Structure

Figure 2 shows the layout of a steam generator model in two-pass configuration. Coal and air enter from the left into the furnace. The combustion heat gets transferred to the furnace walls. As flame temperature is about 1800K, heat transfer is dominated by radiation, while convection is negligible in the first approach.

After combustion, the hot flue gas leaves the furnace and passes several convective heating surfaces. Radiation from the flame to the first convective heating surfaces has to be modeled, as indicated by the arrows in figure 2.

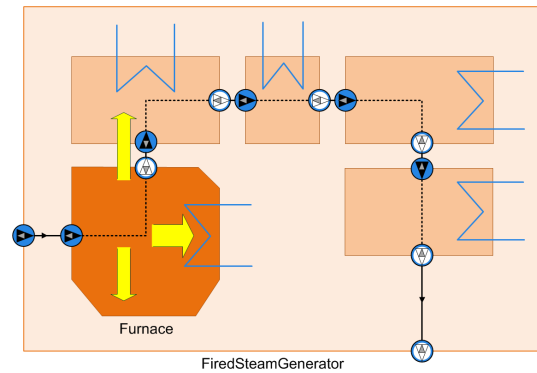


Figure 2: Model layout

Inside the furnace, the non-uniform distribution of mass and heat flows was modeled by spatial discretization in one dimension as shown in figure 3. The size of flue gas elements depends on the geometry of the specific furnace. If more spatial resolution is needed, the zones in figure 3 could be further divided into sub-segments.

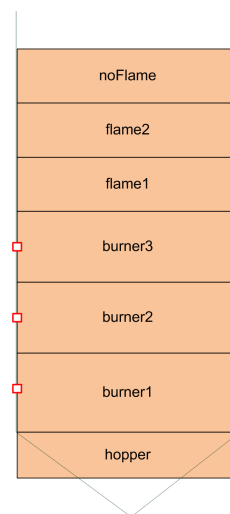


Figure 3: Furnace structure

Each furnace element consists of two components. The equations for gas-particle-flow and combustion are contained in `DispersedFlowZone`. The radiation heat exchange between all surrounding surfaces gets calculated in `RadiationHeatTransfer`.

Figure 4 shows the model diagram of the component `Furnace`. The furnace is separated into several furnace elements according to the levels in figure 3. The left column of components consists of one `DispersedFlowZone` per furnace element, each connected to one `RadiationHeatTransfer` in the right column.

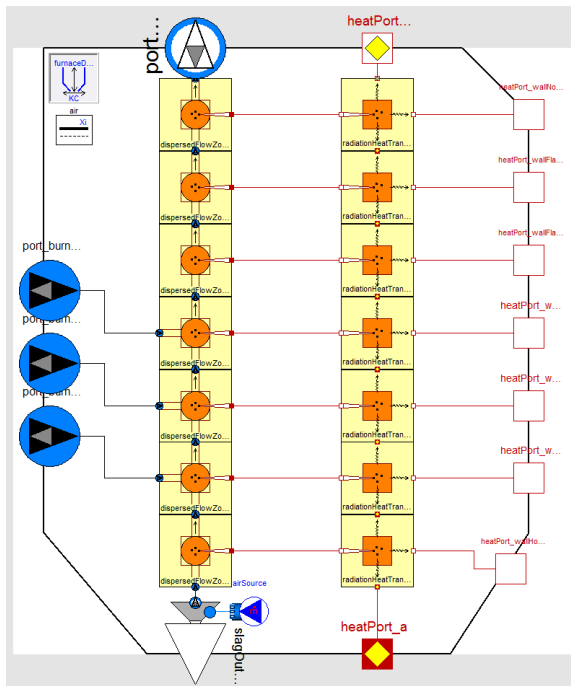


Figure 4: Furnace component diagram

Currently, the heat transfer model is a separate component, leading to a clean model structure. But, because of the strong dependency of the equations, it will be moved into `DispersedFlowZone` as a submodel.

In the presented example, there are three burner levels. Coal enters through the burner levels and gets transported upwards with the flue gas. Small ash particles leave the furnace together with the flue gas, whereas larger ash particles fall into the hopper bottom. The flame at burner levels and in the first section above the furnace is assumed to be non-transparent. Radiation heat can only be exchanged with the furnace walls and the neighboring flue gas zones. After the luminous flame, flue gas zones are partially transparent for heat radiation. Heat can be transferred from the flame to the following gas zones and heating surfaces.

The following sections describe the furnace model in more detail.

3.2 Connectors for Dispersed Flow

In steam power plants there are two major fluid systems, the two phase flow of water/steam and the flue gas. The flue gas in coal fired furnaces contains coal and ash particles which intensify the heat radiation of the flame. Coal particles can travel through multiple flue gas elements and therefore have significant impact on the spatial distribution of heat release. Larger ash particles can move against the direction of flue gas flow because of gravitation. Therefore, new connectors were needed to describe the interaction between flue gas elements.

The `DispersedFluidPort` has been developed in order to have one connector for the flow of gas and solid phase. This minimizes wiring work and corresponds to the physical connections in reality.

The new connector is specialized for the application in the fired steam generator. Hence, the universal approach of `Modelica.Fluid` is not necessary [1, 2]. Well defined flow directions can be assumed and connections between more than two connectors do not have to be handled automatically. Components for splitting and mixing can be provided for the limited number of flue gas branches.

Nevertheless, the `FluidPort` of `Modelica.Fluid` should be used for the gas phase in order to be compatible with existing flue gas zones in `SiemensPower`. The standard components of `Modelica.Fluid` can be used where appropriate (e.g. boundary conditions) and connected to the `FluidPort`.

It was decided to add additional connectors for particles to the `FluidPort` and aggregate all this into the `DispersedFluidPort`. Figure 5 illustrates this approach.

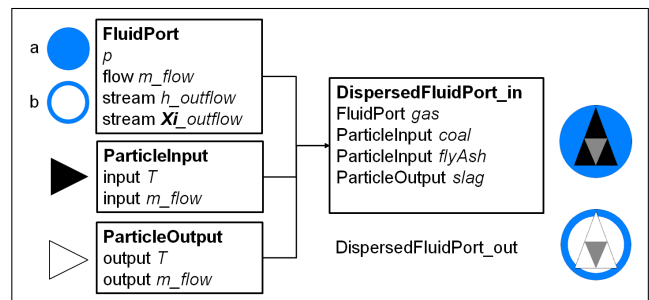


Figure 5: Structure of `DispersedFluidPort`

The Modelica code of DispersedFluidPort:

```
connector DispersedFluidPort_in
  "DispersedFluid input connector"

  // Fluid data
  replaceable package Medium =
    SiemensPower.Media.FlueGas
    "Gas medium model";

  // Ports
  Modelica.Fluid.Interfaces.FluidPort
  gas(redeclare package Medium =
    Medium, m_flow(min=0));

  SiemensPower.Interfaces.ParticleInput coal;
  SiemensPower.Interfaces.ParticleInput flyAsh;
  SiemensPower.Interfaces.ParticleOutput slag;
end DispersedFluidPort_in;
```

ParticleInput and ParticleOutput are causal connectors containing mass flow and temperature as inputs and outputs. A standard Modelica connector with a-causal potential and flow variables was not promising because the coal and ash moves with the flue gas by convection. Therefore it is hard to define a meaningful potential variable which is the driving force of particle flow. More stream variables could have been added to FluidPort for coal and ash but it would not be possible to describe the movement of larger particles against the fluid flow direction.

The use of the aggregated connectors is quite convenient when writing equations with expressions like `port.gas.m_flow` or `port.coal.m_flow`.

3.3 Furnace Element

The furnace element is one element of plug flow of flue gas. The flue gas is modeled as a mixture of ideal gases from Modelica.Media. The following figure shows the mass and energy flows:

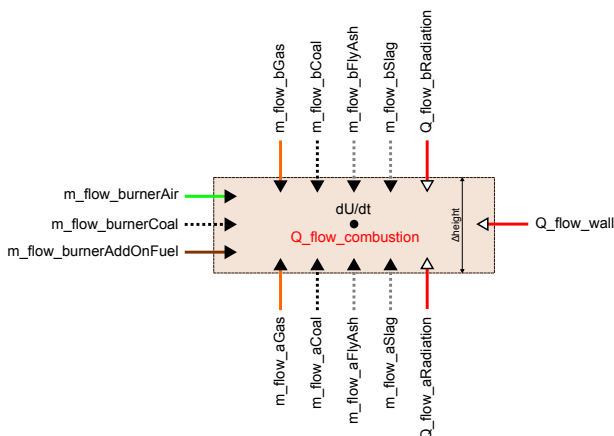


Figure 6: Mass and energy balance of one furnace element

The energy balance of one furnace element:

$$\begin{aligned} \frac{dU}{dt} = & + \dot{H}_{a,\text{flueGas}} + \dot{H}_{a,\text{coal}} + \dot{H}_{a,\text{flyAsh}} + \dot{H}_{\text{burner,air}} \\ & + \dot{H}_{\text{burner,coal}} + \dot{H}_{\text{burner,addOnFuel}} + \dot{H}_{b,\text{slag}} \\ & + \dot{H}_{a,\text{slag}} + \dot{H}_{b,\text{flueGas}} + \dot{H}_{b,\text{coal}} + \dot{H}_{b,\text{flyAsh}} \\ & + \dot{Q}_{\text{combustion}} \\ & + \dot{Q}_{a,\text{radiation}} + \dot{Q}_{\text{wall}} + \dot{Q}_{b,\text{radiation}} \end{aligned} \quad (1)$$

U is the internal energy of the flue gas element. \dot{H} are enthalpy flows, \dot{Q} are energy flows. $\dot{Q}_{\text{combustion}}$ is calculated from the lower heating value and the burned mass flow of coal. The balance of radiation heat transfer and the energy balance result into the correct temperature of the flue gas element.

Radiation heat transfer is calculated according to [3], chapter K. For radiation heat exchange between flame and wall the following equation is used:

$$\dot{Q}_{\text{flame,wall}} = A \sigma \cdot \frac{\varepsilon_{\text{wall}}}{\alpha_{\text{flame}} + \varepsilon_{\text{wall}} - \alpha_{\text{flame}} \varepsilon_{\text{wall}}} \cdot (\alpha_{\text{flame}} T_{\text{wall}}^4 - \varepsilon_{\text{flame}} T_{\text{flame}}^4) \quad (2)$$

$\dot{Q}_{\text{flame,wall}}$ is the heat flow from the flame to the wall, A is the wall area, σ is the Stefan-Boltzmann constant, ε_i are emissivities and α_i are absorbances.

Radiation properties and corresponding surface areas play a key role for correct simulation results. The emissivity of the *wall* is set to an empirical value for each furnace element. The emissivity of the *flame* can be calculated by the model, which uses functions depending on flue gas temperature, gas composition and zone geometry [3].

The standard Modelica heat port was extended by information on emissivity and surface area in order to calculate the net radiation heat flow. The known functions for view factor calculation are implemented in Modelica.

The mass balance is written for every chemical contained in the flue gas and for the three types of particles (coal, fly ash and slag). Momentum balance is reduced to a hydraulic resistance of the gas phase until more data is available.

3.4 Usability Aspects

When using and maintaining the described model, three aspects could be improved. Handling of 3D ge-

ometry parameters is not intuitive in 2D model diagrams. All geometry data is stored centralized in a record `FurnaceData`, but the user has to make sure the correct parametrization of the furnace elements. Some 3D-support for the parametrization of 1D-models would simplify this task.

It is not easy to implement the radiation heat balance in component oriented model where data exchange between objects should be transparent for the user. Unfortunately, there is no simple solution for 3D radiation exchange.

There is no language or tool support for 1D spatial discretization of partial differential equations in Modelica. For-loops of equations or connects are written, without graphical representation. The model user mostly has to look into the equations in order to follow the details of discretization. Possibly annotations could be generated to visualize the discretization.

4 Results

In order to test the furnace model, a test environment with ideal components for heat controller, coal feeder, coal pulverizer, coal distributor and precipitator has been developed (figure 7). Temperature boundary conditions are connected to the furnace instead of water tubes. The components of this test environment can be configured and improved according to the specific simulation task.

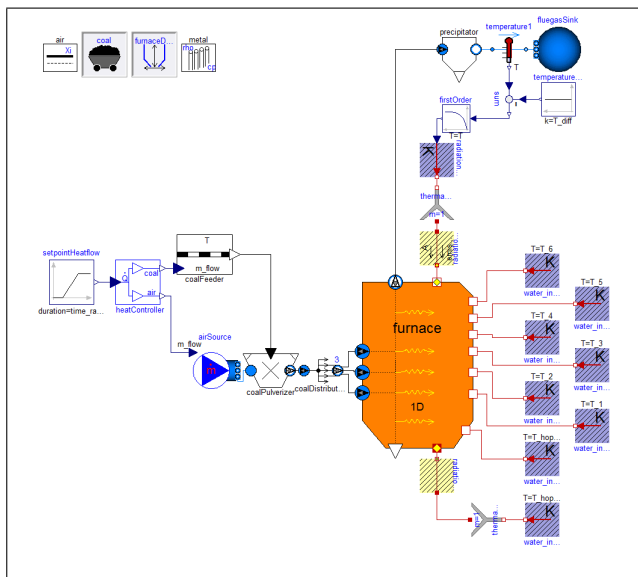


Figure 7: Test environment of furnace model

The practicality of the model concept was tested by calculations on an existing large scale power plant. A model of this plant was available in an in-house steady state tool. Parameters and boundary conditions of the Modelica model were obtained from construction data of the plant and the existing model. The results of the existing model were compared to steady state results of the new Modelica model. Figure 8 and figure 9 show these results. The rectangles on the height coordinate indicate the three burner levels. The solid line shows the Dymola result which agrees with the dashed reference line in the lower segments. At the furnace exit, the fit is not that good because of the imprecise test environment without convective heating surfaces.

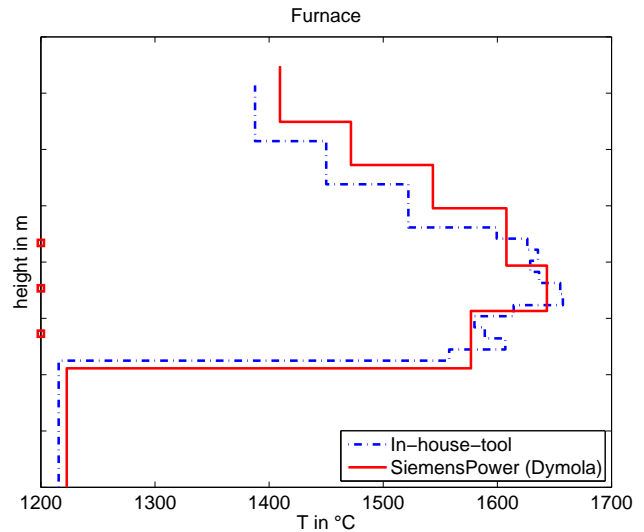


Figure 8: Temperature versus furnace height

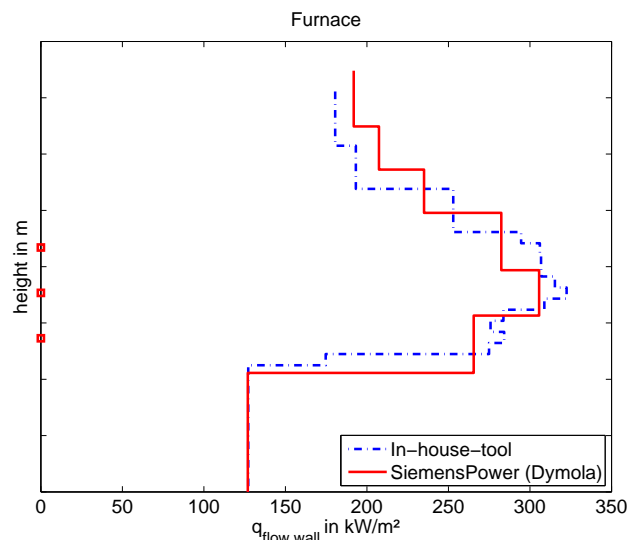


Figure 9: Heat flux versus furnace height

5 Conclusion and Outlook

A modeling framework for coal fired furnaces was described which can be used for dynamic simulations of steam power plants. The new Modelica components serve as an extension to the existing SiemensPower library.

Further validation is needed for transient operation. The next step would be to attach evaporator tubes and super-heaters in order to compare the model to dynamic simulation results and measurements.

References

- [1] Franke, Rüdiger ; Casella, Francesco ; Otter, Martin ; Sielemann, Michael ; Elmqvist, Hilding ; Mattson, Sven Erik ; Olsson, Hans: *Stream Connectors – An Extension of Modelica for Device-Oriented Modeling of Convective Transport Phenomena*. In: Proceedings of the 7th International Modelica Conference, September 20th-22nd, 2009, Como, Italy, p. 108-121.
- [2] Franke, Rüdiger ; Casella, Francesco ; Sielemann, Michael ; Proelss, Katrin ; Otter, Martin ; Wetter, Michael: *Standardization of Thermo-Fluid Modeling in Modelica.Fluid*. In: Proceedings of the 7th International Modelica Conference, September 20th-22nd, 2009, Como, Italy, p. 122-131.
- [3] VDI-Gesellschaft Verfahrenstechnik und Chemieingenieurwesen (Ed.): *VDI-Wärmeatlas*. Ed. 10. Berlin : Springer, 2006.
- [4] Gall, Leo: *Dynamic Modeling of Fired Steam Generators*. Erlangen, Germany: Diploma thesis, Technische Universität München, 2010.
- [5] Dassault Systèmes AB: *Dymola Version 7.4*.
<http://www.dymola.com>
- [6] Spliethoff, Hartmut: *Power Generation from Solid Fuels*. Berlin : Springer, 2010.

Inclusion of Reliability and Safety Analysis Methods in Modelica

Christian Schallert

German Aerospace Centre (DLR), Institute of Robotics and Mechatronics
 82234 Wessling, Germany
 christian.schallert@dlr.de

Abstract

A method is developed to combine techniques of reliability and safety analysis with the Modelica language, which is now widely used for the modelling and simulation of technical systems.

The method allows to perform a reliability or safety analysis on the system model that is created and used for simulation studies. The procedure automatically determines the so called minimal path sets or minimal cut sets of a system, its failure probability and critical components.

The reliability and safety analysis methods are incorporated in a Modelica library that is established for the modelling and simulation of aircraft on-board electrical power systems. The recent trend towards a broader use of electric system technologies on commercial aircraft has motivated the creation of this kind of model library, which supports the conceptual design and optimisation of on-board electrical systems regarding power behaviour, weight, reliability and safety.

Keywords: reliability; safety; fault modelling; redundant system; minimal path sets; minimal cut sets

1 Introduction

Much of the information needed for reliability or safety analysis is contained already in complex system models that are usually built in Modelica. The specific modelling additions needed, as well as the concept of an automated reliability and safety analysis procedure are described in this paper.

The analysis procedures evaluate the physical behaviour of a system model in multiple simulations. Representing not only the normal but also the faulty behaviour of components is needed as an addition to the modelling, as described in section 2.1. Section 2.5 illustrates the scope and method of the reliability and safety analyses and their relevance regarding aircraft on-board systems. A way of minimising the involved computational effort is outlined in 2.5.4.

Then, section 3.1 presents an example model of an electric power system of a recent large commercial aircraft. Subsequently, a safety and reliability analysis are conducted on the model for example scenarios, the results of which are graphically presented and discussed in sections 3.2 and 3.3.

2 Modelling Approach and Outline of Reliability and Safety Analysis Method

2.1 Component Fault Modelling

A variety of object-oriented model libraries has been developed in the Modelica language, as generally known. Typically, each component model contains a description of the normal operational behaviour by differential and/or algebraic equations.

For the purpose of reliability and safety analysis, the component models have to be enhanced to describe also the failure behaviour by physical equations. Basic examples are given hereafter by the model approach taken for some common electric components.

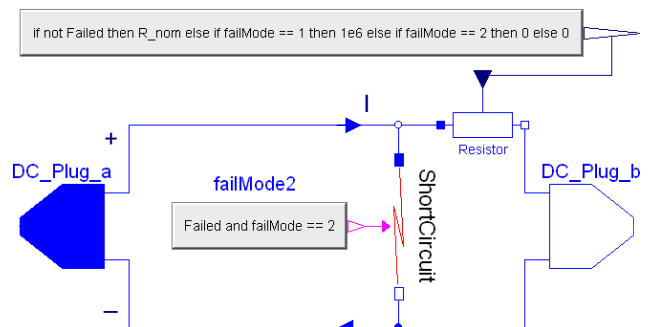


Figure 1: Modelica model of an electrical cable with normal and failure behaviour

A two-core electric cable can be described as an ohmic resistor. For the normal function of the cable, its resistance R is in the order of $R_{nom} \approx 10^{-1} \Omega$. An open circuit (O/C) failure of the cable is characterised by a very large resistance, e.g. $10^6 \Omega$, whereas a short circuit failure (S/C) can be described by small resistance

of $10^{-5} \Omega$ connecting the two cores of the electric cable. As can be seen in Figure 1, the resistor element used to model the short circuit is always present in the cable model, but it has a large value of $10^6 \Omega$ in cases other than a short circuit failure.

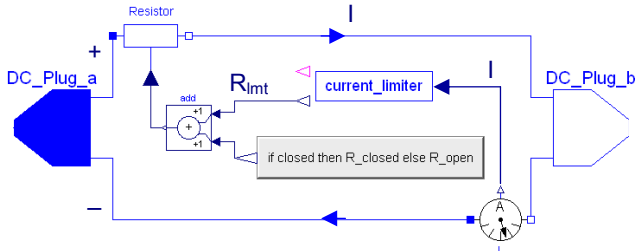


Figure 2: Electric contactor model with normal and failure behaviour and current limitation function

An electrical contactor is also modelled as a variable ohmic resistor, as shown in Figure 2. The opening and closing of the contactor is described by different resistance values, $R_{open} = 10^6 \Omega$ for the open and approximately $R_{closed} = 2 \cdot 10^{-3} \Omega$ for the closed contactor state. The small resistance of the closed contactor effects a voltage drop, which can be specified by a model parameter. Optionally, this model may be used as a current limiting device, similar to a circuit breaker. Figure 3 shows the current limitation function: By increasing the resistance value R_{limt} above zero, the limitation function prevents that the actual current, denoted by I , exceeds the nominal current I_{nom} . This kind of model has been selected, since it does not require any resetting after the limitation function has been activated in the simulation, other than a real circuit breaker which must be reset after having tripped. For a contactor or circuit breaker, there are a couple of conceivable failure modes, and the two most relevant of them are described in the model: An open circuit failure, which is modelled in the same manner as for the electrical cable, and a fails to open malfunction. The latter failure mode means a loss of the current limitation function, i.e. failure to protect against overcurrent.

As the examples suggest, a DC modelling approach has been selected for the electrical components. A single or three phase AC component is described by the equivalent DC component with root mean square values for voltage and current. A three phase AC component is represented by a single phase, assuming that the three phases are symmetrical. Thus, the substitute single phase generates, conducts or uses a third of the entire current and power. Furthermore, the electrical behaviour is described by algebraic physical equations for the normal and several failure modes of each component; differential equations are

omitted for simplification. This is judged as adequate regarding the objective of performing network architecture level conceptual design and optimisation, including the analysis of steady-state electric power behaviour, reliability, safety and weight.

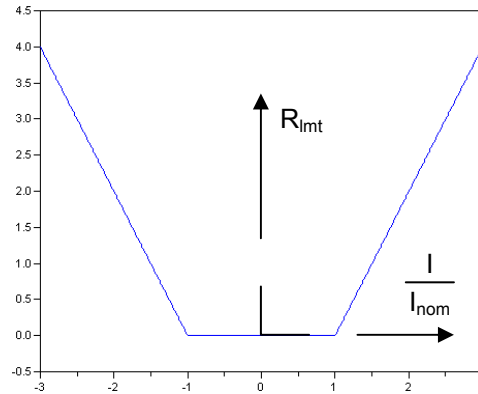


Figure 3: Current limitation function

Each component model has a boolean input signal to control its status, i.e. normal operation, failure mode 1, failure mode 2 etc., as applicable. The status can be shifted during simulation. The failure probability $p_i = 1 - e^{-\lambda_i \cdot t}$ is stored in each component model as a changeable parameter. Constant failure rates λ_i and exponentially distributed lifetimes are a common assumption in reliability and safety analysis.

The weight of a component is given dependent on sizing parameters of the accordant component model, such as the weight of a generator depends on its nominal power and speed.

Thus, a Modelica library of electric component models, that are augmented with a basic failure behaviour and parameterised weights, is developed. In doing so, the concept of creating component models that are usable regardless of the application or physical context, is being followed. Compatibility with existing model libraries is maintained.

2.2 Concept of Model Library with Included Analysis Procedures

The introduced Modelica library of electric component models and accompanying procedures for automated electric loads, reliability and safety analysis forms a tool for the conceptual design of aircraft on-board electric power systems. The tool is named as the Electrical Network Architecture Design Optimisation Tool - ENADOT. Besides reliability and safety, ENADOT is prepared to evaluate electric network architecture concepts w.r.t. to power behaviour and weight, as illustrated by Figure 4.

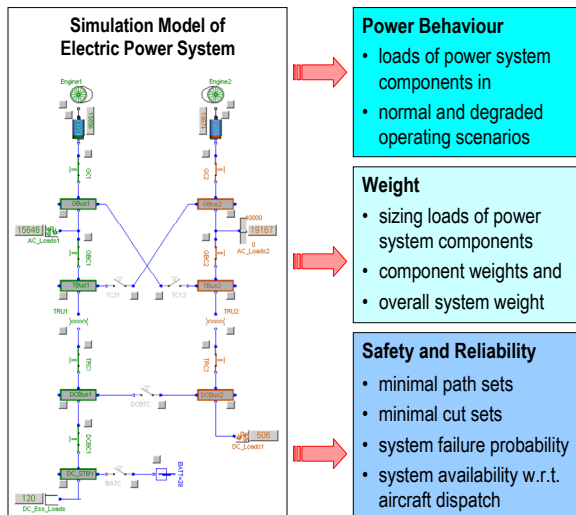


Figure 4: Concept of a Modelica-based tool for electric network architecture design analysis and optimisation - ENADOT

Large and complex models of electrical networks can be composed using the graphical editor of Dymola in the known manner.

Aircraft on-board electrical power systems are of particular interest w.r.t. reliability and safety, since they supply various loads, many of which fulfill a function that is essential for the safety of flight. Due to the recent trend to replace hydraulic and pneumatic supply systems by electric power on-board of commercial aircraft, the electric demands and thus the overall size of the electric system tend to grow. This increase of size and complexity also calls for a comprehensive modelling and simulation tool, due to the limitations of traditional design methods.

2.3 Modelling of Electric System Operating Modes

Electric power systems on-board of aircraft are typically split into several independent channels for redundancy, each comprising an engine driven generator, a distribution network and a number of loads. If failures occur, the network is reconfigured automatically to isolate the fault and to secure power supply to most of the loads, with priority to the essential ones. This reconfiguration capability has to be built into the system model accordingly. It is achieved by including the open / close logics of the various electric network contactors, which link (or cut off) the generators, busbars and loads. Thus, an electric network architecture model can be simulated for a normal and various abnormal operating scenarios.

2.4 Visualisation of System Operation and Interactive Checking

The diagram layer of an electric network architecture model is used also for dynamic and graphic display of the open / closed states of the various contactors, as well as of the resulting flow of electric power by different colours. For this purpose, ENADOT employs the visualisation and real-time simulation capabilities of Dymola and Visual C++.

If a component is energised, i.e. under voltage or conducting current, then its shape is coloured, as shown in Figures 5 and 6. The accordant colour stems from the generator or battery which energises the component. Passive components are shown in grey colour. The user can interactively shift the operating / fault modes of the electric network components, i.e. inject failures by mouse-click, and observe the resulting system behaviour by the visualisation in the diagram layer. That way, the model implementation of an electric network architecture is readily verified with regard to the intended behaviour.

2.5 Automated Analysis Procedures

To evaluate an electric network architecture model, ENADOT provides functions for an electric loads analysis, computation of component weights and overall system weight, a safety analysis which examines the probability of failure of voltage supply to a single or several busbars, a reliability analysis which evaluates the operational availability (→ aircraft dispatch reliability) of an electric network architecture, as well as compilation of a bill of material. The electric loads and safety analyses rely on the capability of an electric network model to simulate various operating modes and to bypass failed components.

These procedures are written as Modelica functions in algorithm syntax. They are part of the ENADOT library and simply rely on Dymola for execution.

2.5.1 Electrical Loads

The electric loads analysis determines the highest electric power generated or carried by a component in the most adverse operating case. To compute the highest electric power (design point) of any component of an electric network model, the function simulates it automatically for normal and degraded operating scenarios. As a result, the design point is provided for each component combined with its temporal occurrence during a flight cycle. Then, the sizing parameters of each component are selected which in turn yields the component weights and the overall weight of an electric network architecture.

2.5.2 Safety

By means of a safety analysis function embedded in ENADOT, the probability of loss of voltage supply to a single or several busbars of the electric network can be computed. Analysing the probability of loss of voltage supply to busbar(s) is particularly relevant if electric loads are connected to them that perform a function which is critical regarding the safety of flight. The scenarios to be investigated have to be supplied by the operator, e.g. “system is functional if at least one DC busbar is energised” or “system has failed if voltage is lost on the AC essential busbar”.

Before starting the self-acting safety analysis, the operator can choose between the block-diagram (RBD) or the fault tree analysis (FTA) method.

The former is based on the identification of minimal path sets: A minimal path set is a combination of intact components that causes a system to be functional in the sense of the specified scenario, e.g. “at least one DC busbar energised”. Minimal means that a path set contains only as many intact components as are necessary for the system to be functional. Redundant systems are characterised by the existence of several minimal path sets for a specified scenario, e.g. several ways of energising a busbar. By nature, a minimal path sets analysis considers only two states per component: intact or, respectively, failed [2].

The fault tree analysis method corresponds with the determination of minimal cut sets: A minimal cut set is a combination of defective components, which causes the system to fail in the opposite sense of the specified scenario, e.g. “no single DC busbar energised”. Here, minimal means that a cut set consists of only as many defective components as causes the system to fail. Minimal cut sets comprise one (1st), two (2nd) or three (3rd order) defective components, and the probability of occurrence of a minimal cut set decreases rapidly with the number of components that belong to it. A redundant electrical system, in turn, is characterised by the fact that no 1st order minimal cut sets exist, apart from own defects of the busbar under consideration, but rather combinations of two or three defective components lead to the loss of voltage on busbar(s) and hence system failure.

Furthermore, the minimal cut sets analysis differs from the minimal path sets analysis by the consideration of all possible states of each component (intact, failure mode 1, failure mode 2, etc.). It is thus more complex and computationally more intensive than the minimal path sets analysis. The result, though, is equivalent to that of the established method of fault tree analysis, which is generally accepted as a verification of system safety. The computationally less intensive minimal path sets analysis provides quicker

available results, which are normally used as a first estimate of system safety in the design process.

It must be noted that a model-based safety analysis only covers phenomena captured in the scope of modelled physics (section 2.1). Though the analysis is exhaustive to this extent, it is up to the designer to regard other possible threats, e.g. common causes such as humidity or electromagnetic interference.

The key definitions regarding safety analysis based on minimal path and minimal cut sets are as follows.

The common assumption of exponentially distributed lifetimes of the components c_i means component failure rates λ_i that are constant over lifetime. Thus, the probability of a component failure is

$$p_i(t) = \begin{cases} 1 - e^{-\lambda_i t}, & t \geq 0 \\ 0, & t < 0 \end{cases}$$

The probability of a minimal path set to occur is

$$P(MP) = \prod_{c_i \in MP} (1 - p_i), \text{ with the components } c_i \text{ and}$$

the individual failure probabilities p_i . Likewise, the probability of occurrence of a minimal cut set is

$$P(MC) = \prod_{c_i \in MC} p_i$$

The probability of system operation can be computed from m detected minimal path sets as

$$\begin{aligned} P_{\text{system-operation}}(p_i) &= P(MP_1 \vee MP_2 \vee \dots \vee MP_m) \\ &= \sum_{j=1}^m (MP_j) - \sum_{i=1}^{m-1} \sum_{j=i+1}^m P(MP_i \wedge MP_j) + \dots \\ &\quad + (-1)^{m+1} P(MP_1 \wedge MP_2 \wedge \dots \wedge MP_m) \end{aligned}$$

Likewise, the probability of system failure can be calculated from n detected minimal cut sets as

$$\begin{aligned} P_{\text{system-failure}}(p_i) &= P(MC_1 \vee MC_2 \vee \dots \vee MC_n) \\ &= \sum_{j=1}^n (MC_j) - \sum_{i=1}^{n-1} \sum_{j=i+1}^n P(MC_i \wedge MC_j) + \dots \\ &\quad + (-1)^{n+1} P(MC_1 \wedge MC_2 \wedge \dots \wedge MC_n) \end{aligned}$$

Generally, the relation between the probability of operation and failure, for a single component or a complex system is $p_{\text{failure}}(t) + p_{\text{operation}}(t) = 1$

Since the computation of system operation or failure probability from the above Poincaré formula can lead to a very large number of products, algorithms for sums of disjoint products [4] have been developed to reduce the size of the formula and to facilitate its numerical evaluation.

An analysis example can be viewed in section 3.2.

2.5.3 Reliability

Whereas a safety analysis is focused on failure events that can be critical with respect to the safety of flight, reliability is concerned with the operational availability of a system or entire aircraft. Operational availability or dispatch reliability is a measure for the likelihood of an aircraft fulfilling its mission, that is for a commercial aircraft to make revenue flights on time with passengers and/or cargo. More precisely, the dispatch reliability is defined as the percentage of scheduled flights which depart without having a delay of more than 15 minutes due to technical reasons, or a cancellation [1].

The commercial pressures have instigated the ability to continue to dispatch an aircraft with given system faults. Redundant design of aircraft on-board systems is adopted not only to fulfill the safety requirements, but also for reasons of dispatch reliability. In turn, this requires to examine the ability of a degraded system, when one or more failures have already occurred, to meet the safety requirements.

ENADOT has an embedded function, developed in the Modelica language, for computing system operational availability. It is in essence a minimal path sets analysis, which uses information about allowed component deficiencies, so called MEL-items (Minimum Equipment List), entered by the operator.

Let C_{system} be the set of all system components c_i , C_{MEL} be the given set of n_{MEL} possibly defective components and k the number of intact components required, $k \leq n_{MEL}$ and $C_{MEL} \subseteq C_{system}$. Then,

$m = \binom{n_{MEL}}{k}$ minimal path sets are generated with

the following properties:

Each minimal path set MP_1, MP_2, \dots, MP_m contains those system components c_i that are not an MEL-item, i.e. not part of the set C_{MEL} .

$$c_i \in C_{system} \setminus C_{MEL} = \{c_i \mid (c_i \in C_{system}) \wedge (c_i \notin C_{MEL})\}$$

As well, k intact components from the set of MEL-items C_{MEL} are included in each minimal path set.

E.g. for $k = 2$ and $n_{MEL} = 3$, the $\binom{3}{2} = 3$ following

minimal path sets are composed:

$$MP_1 = C_{system} \setminus C_{MEL} \cup \{c_1, c_2\}$$

$$MP_2 = C_{system} \setminus C_{MEL} \cup \{c_1, c_3\}$$

$$MP_3 = C_{system} \setminus C_{MEL} \cup \{c_2, c_3\}$$

An analysis example is provided in section 3.3 for the electric power system introduced by section 3.1.

2.5.4 Minimising the Computational Effort Involved with Safety Analysis

Analysing the effect of combinations of intact and failed components on the occurrence of system function or failure can lead to an exponential growth of combinations to test. Regarding the detection of minimal path sets, 2^n possible states would have to be evaluated for a system of n components, e.g. $2^{20} > 1 \cdot 10^6$ for $n = 20$.

To avoid the unfeasibility of automated analysis caused by an excessive amount of system states to test, strategies are developed to exclude inapplicable combinations of intact / failed components from the procedure.

The minimal path sets analysis procedure of ENADOT draws on two kinds of information contained in a system model. In a first step, the object structure of the system model, i.e. the arrangement of components and connections, is evaluated. Advantage is taken of the fact that the structure of object-oriented models is similar, although not exactly identical with minimal path sets. Regarding the object-oriented model structure as a graph, an adapted depth-first search algorithm is used to find a moderate number of candidates of minimal path sets.

In a second step, the candidates are checked by simulating the system model accordingly, to eventually extract the minimal path sets from the amount of candidates. This two-stage approach – depth-first search and then simulation – considerably reduces the overall computation effort, leading to a procedure that is viable even for systems of a size as shown in section 3.1.

After the minimal path sets of a system have been determined for a given scenario, the probability measures are computed as described in section 2.5.2.

For the minimal cut sets analysis, the theoretically possible number of system states is even higher: Assuming that three states (intact, failure mode 1, failure mode 2) have to be considered for each component of a system, this would lead to 3^n possible system states, e.g. $3^{20} > 3 \cdot 10^9$ for $n = 20$.

Here, the strategy of minimising the amount of system states to check includes at first to determine the minimal path sets, as described above. Then, minimal cut sets are searched for according to heuristic rules that draw on the position of components in the system and their modes of failure. For instance, only combinations of failed components that belong to different minimal path sets or which are located adjacent to a minimal path set, are checked.

3 Modelling and Analysis Case Study

This section illustrates the capabilities of ENADOT with respect to safety and reliability analysis by the example model of an aircraft on-board electric power system. Figure 5 shows the model, the basic structure and characteristics of which are oriented to the electric power system of the Airbus A380. The model complies with the typical configuration and functionality of electrical systems of this aircraft category, and it is thus adequate for a demonstration of the scope of ENADOT. The model has been developed based on a description, conceptual sketch and listing of the key electrical loads, which have been found in section 5.12.1 of reference [5]. It may differ in some minor respect from the actually built and flying electric system of the A380, yet, this does not affect the description of the scope of ENADOT.

3.1 Electric Power System Modelling Example

The schematic shown in Figure 5 is a direct snapshot of the electric power system model. It includes the following, salient components and features:

- four engine driven 3-phase 115 VAC / 150 kVA Variable Frequency (VF) generators, identified as G1, G2, G3 and G4
- two 3-phase 115 VAC / 120 kVA Constant Frequency (CF) generators, driven by the Auxiliary Power Unit (APU), denoted as AG1 and AG2
- a 70 kVA Ram Air Turbine (RAT) driven emergency generator, named RatG
- three 300 A Battery Charger Regulator Units (BCRU) – these are regulated Transformer Rectifier Units (TRU) – named EssBCRU, BCRU1 and BCRU2
- a 300 A TRU identified as APU_TRU
- four 28 VDC batteries, denoted as ESS_BAT, BAT1, BAT2 and APU_BAT
- a static inverter, named INV, for emergency supply of the AC_EMER busbar

3.1.1 System Functionality

Figure 5 shows the normal in-flight operation of the electric power system. As can be seen, each engine driven generator G1 (blue), G2 (green), G3 (magenta) and G4 (bronze) energises its associated busbar AC_1, AC_2, AC_3 and AC_4. The two APU driven generators AG1 (purple) and AG2 (yellow) are available, but not engaged. If a generator fails, the neighboured generator will take over by closing the ACTC1 or ACTC5 contactor. If both generators

on one side fail, then cross-transfer through the ACTC2, 3 and 4 contactors will sustain all AC busbars energised, with yet decreased overall available power. Split generator operation is maintained in all cases since the engine driven AC generators are variable frequency, each dependent on the speed of the related engine.

The AC buses supply the non-essential cabin loads Galley1, 2, 3 and 4 and In-Flight Entertainment (IFE) 1 and 2. These form an intermittent load of up to ~320 kVA (~80 kVA per galley including cooling) and ~60 kVA (IFE). Those AC loads that are vital for the safe operation of the aircraft are connected to the AC_ESS and AC_EMER busbars. These are airspeed probes and windshield heating, as well as motor driven hydraulic pumps and a set of Electro-Hydrostatic flight control Actuators (EHAs) needed to maintain a minimum acceptable level of airplane controllability. The AC essential loads sum up to ~60 kVA. The AC_ESS and AC_EMER busbars are supplied either by the AC_1 busbar (normal case) or, if AC_1 fails, from the AC_4 busbar. Should all engine generated power fail, then the RAT driven generator RatG can accept the AC_EssLoads and AC_EmerLoads. The latter can also be powered by battery through the static inverter INV, e.g. during RAT transit.

Other than the AC part of the electric power system, the 28 VDC part offers a no-break power capability even during changes of system status, which is crucial to the functioning of vital control systems, such as engine and flight control computers, avionics systems, flight deck instruments and radio communication. These loads are represented in the model by DC_EssLoads, DC1_Loads and DC2_Loads and account for ~4 kW altogether. The cabin lights make up ~15 kW of power, supplied by the non-essential part of the DC system.

3.1.2 Degraded System Operation

Figure 6 shows the electric power system in a conceivable mode of degraded operation. The failed components Engine1, G2, APU and BCRU2 are marked in red colour. Since the power supply from G1 and G2 is lost, the AC_1, AC_2, AC_3, DC_1 and DC_2 busbars are energised by G3 (magenta). Failure of the BCRU2 has been recovered by closing the DCTC2 contactor. The other remaining generator G4 (bronze) energises the AC_4, as well as the essential busbars AC_ESS, AC_EMER and DC_ESS. As the scheme also shows, half of the cabin loads – galleys, IFE and lights – have been suspended, whereas the essential loads remain fully satisfied.

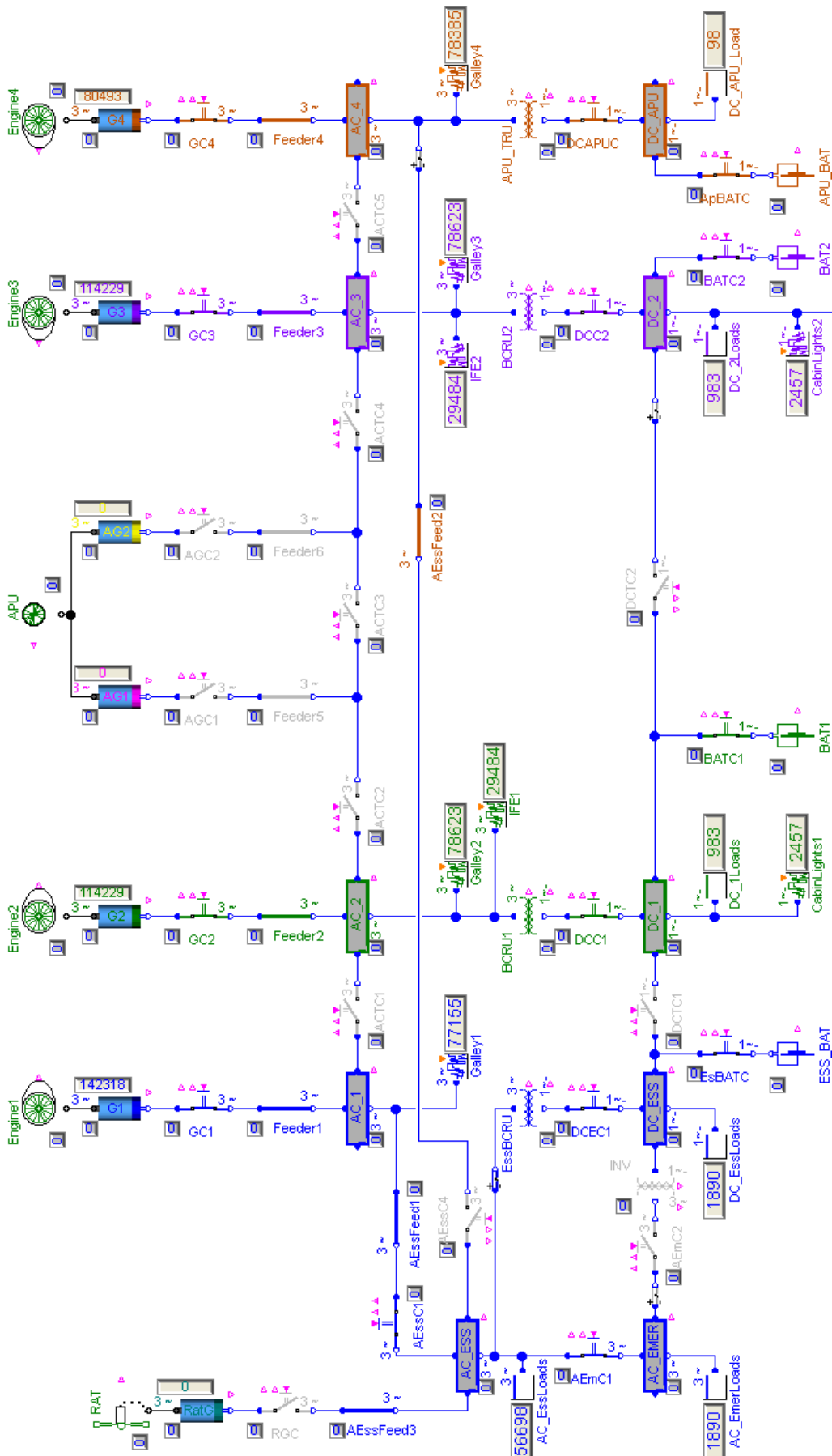


Figure 5: Electric network model of a recent four-engine long range aircraft, scheme shows normal operation in flight

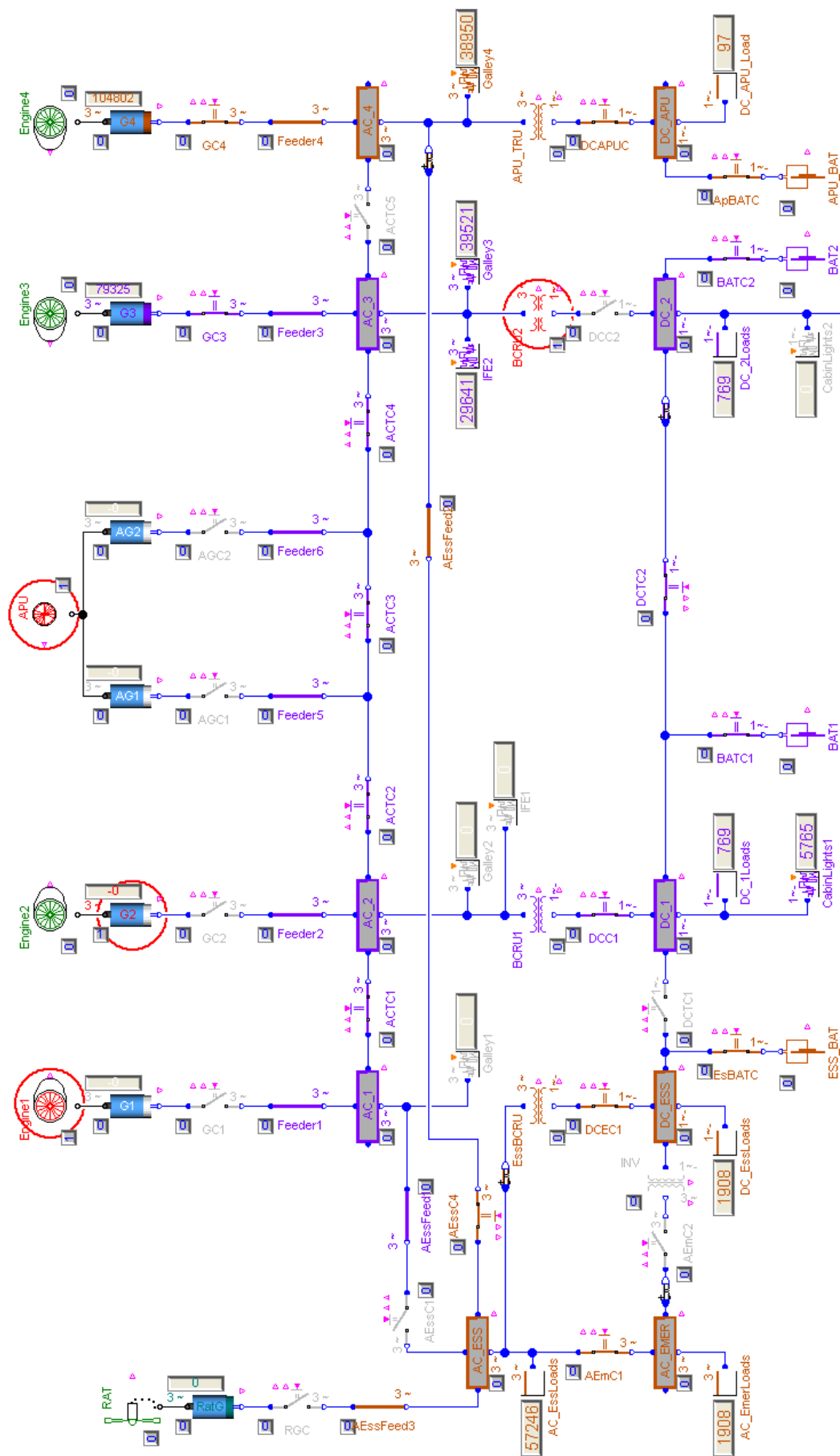


Figure 6: Electric network model of a recent four-engine long range aircraft, scheme shows degraded operation in flight after Engine1, Generator2, APU and BCRU2 failure

3.2 Safety Analysis Example Result

This section shows the result of a safety analysis conducted for the supply to the AC_1 busbar of the introduced electric system. This non-essential busbar has been selected to serve as an example, since the result is relatively compact.

Figures 7 to 12 show the six determined minimal path sets. They are depicted graphically and directly in the model diagram, after completion of the analysis procedure. Components belonging to a minimal path set appear in the colour of the connected generator, failed components in grey. In normal operation, AC_1 is supplied by generator G1 (Figure 7), which can be transferred to another engine or APU driven generator in abnormal operating cases. Hence,

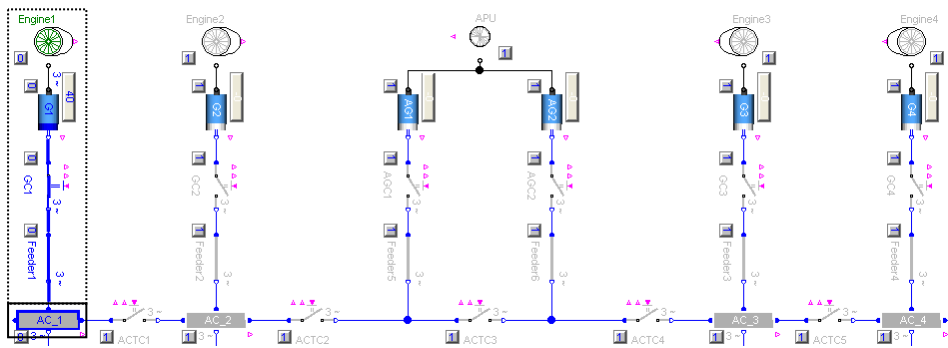
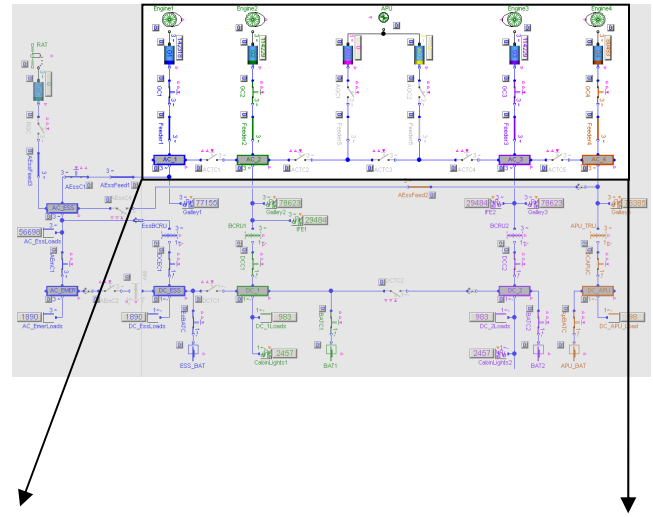


Figure 7: Minimal path set 1 - AC_1 busbar energised by G1

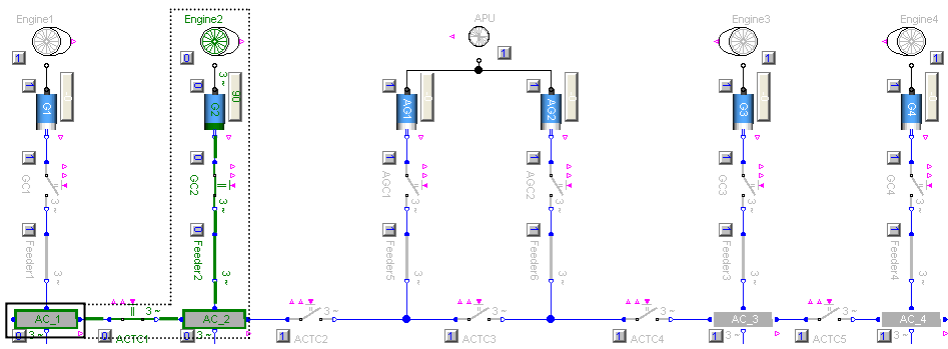


Figure 8: Minimal path set 2 - AC_1 supplied by G2 across AC_2

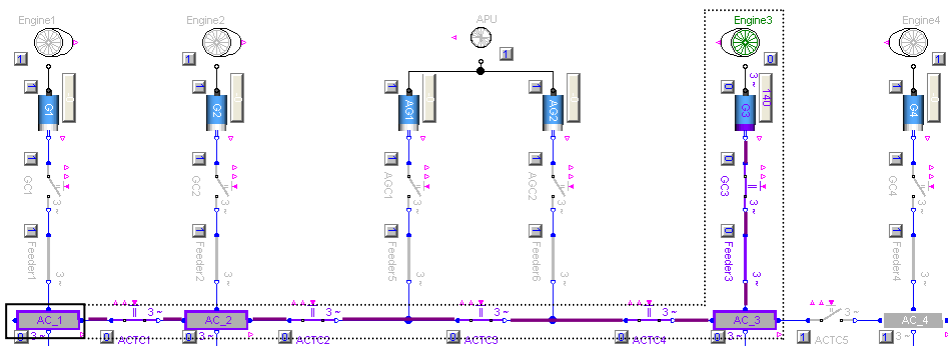


Figure 9: Minimal path set 3 - AC_1 fed by G3 through AC_3 and AC_2

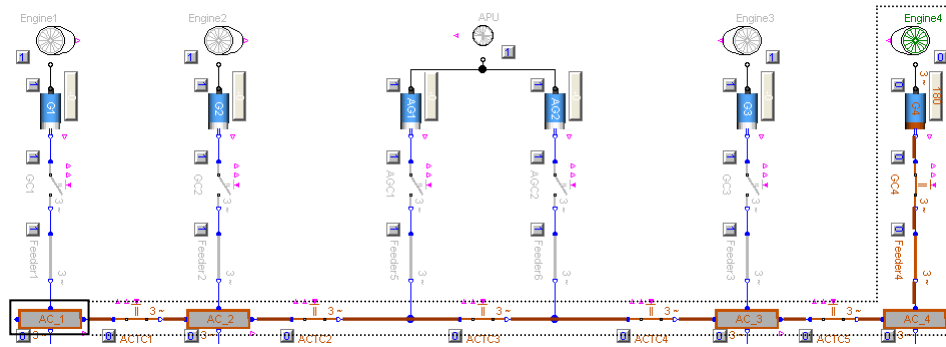


Figure 10: Minimal path set 4 - AC_1 energised by G4 across AC_4, AC_3 und AC_2

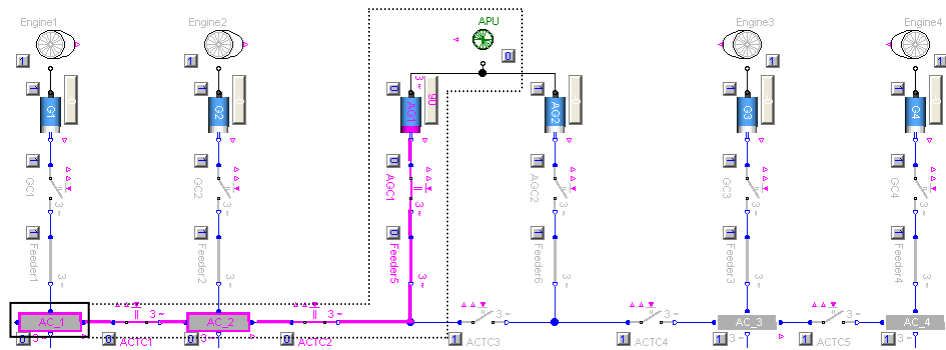


Figure 11: Minimal path set 5 - AC_1 fed by AG1 through AC_2

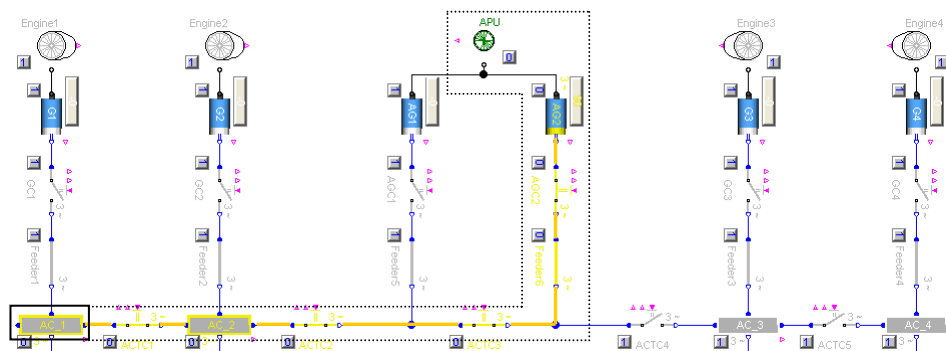


Figure 12: Minimal path set 6 - AC_1 supplied by AG2 across AC_2

AC_1 has multiple redundancy as to voltage sources. The minimal path sets analysis accounts for the connections of components, their potential faults and network reconfiguration logics. The result, e.g. the six minimal path sets found for the AC_1 busbar, is thus also a check of the correct functioning of the system and its implementation as a model.

As explained, two states are considered for each component, intact or failed, in the minimal path sets analysis. Many components yet have two or more failure modes. Amongst others, the following are realised in the modelling: “open circuit” and “short circuit” for a cable or a busbar, “open circuit” and “fails to open” for a contactor, “loss of output voltage” for a generator. The minimal cut sets analysis accounts for every failure mode of all components and the resulting effects on the electric network.

A total of 5 first order and 21 second order minimal cut sets were identified by the analysis procedure and are listed in Table 1. Figures 13 to 19 show typical cases for the scenario “loss of voltage on AC_1”.

Besides own possible faults of the AC_1 busbar – open circuit (Figure 15) or short circuit – other single component faults exist that lead to a loss of voltage on AC_1: e.g. a short circuit of cable Feeder1 (Figure 14), which is directly connected to the busbar, or in the same manner a short circuit of cable AEss-Feed1 (Figure13). Typical examples of 2nd order minimal cut sets are a failure of a component that feeds AC_1 in normal system operation, in combination with another component failure that prevents cross-transfer through AC_2 and ACTC1. Examples can be viewed in Figures 16 and 18.

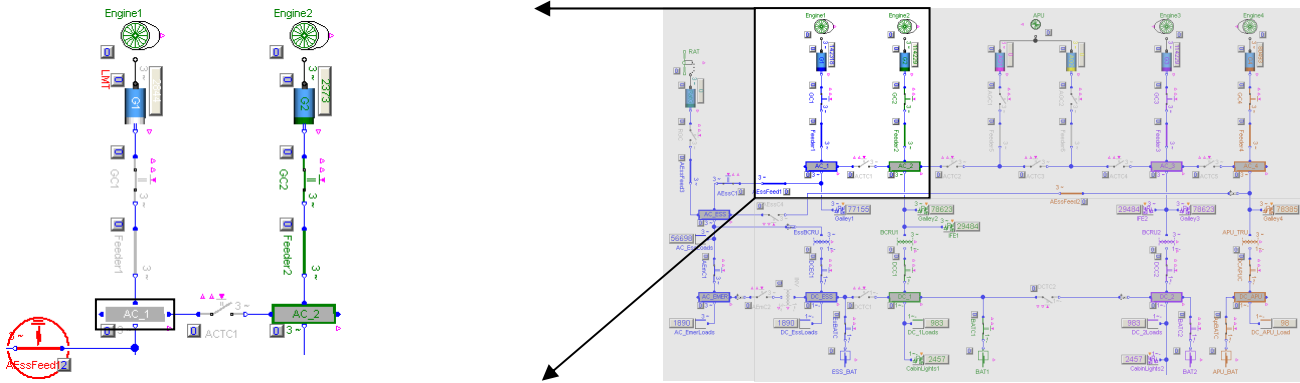


Figure 13: Minimal cut set 1-2: short circuit of cable leads to loss of voltage on AC_1 busbar

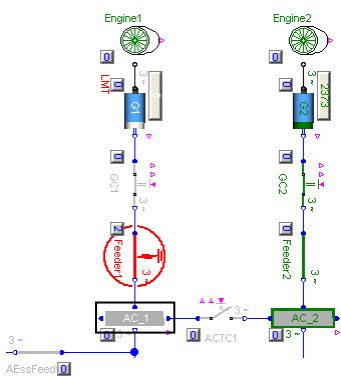


Figure 14: Minimal cut set 1-3: cable short circuit causes failure of AC_1

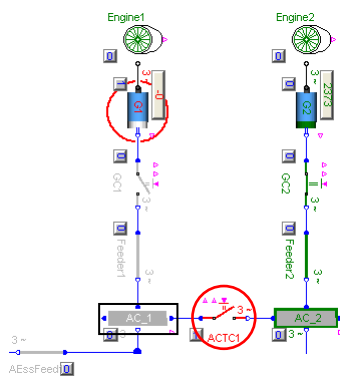


Figure 16: Minimal cut set 2-7: G1 fault and open contactor cause AC_1 failure

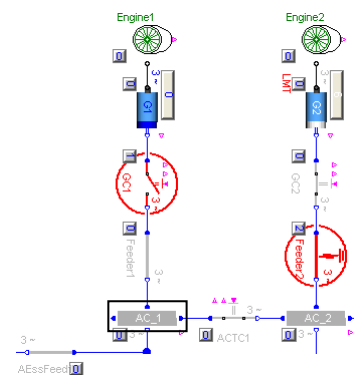


Figure 18: Minimal cut set 2-5: open contactor and shorted cable lead to AC_1 failure

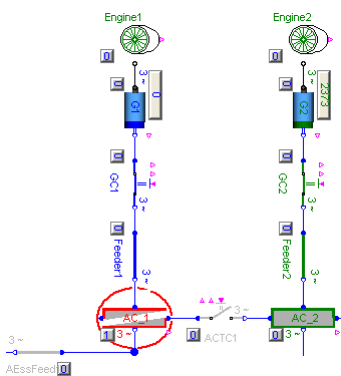


Figure 15: Minimal cut set 1-5: open circuit leads to loss of AC_1

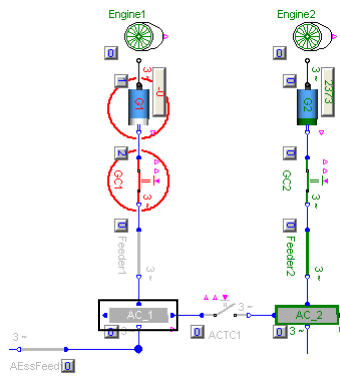


Figure 17: Minimal cut set 2-14: G1 fault and stuck closed contactor lead to loss of AC_1

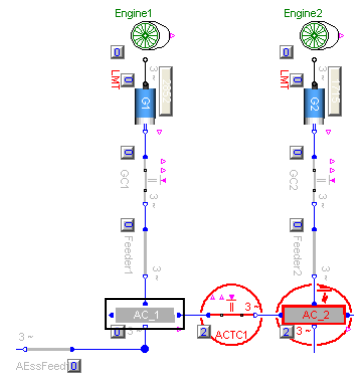


Figure 19: Minimal cut set 2-21: short circuit of AC_2 and stuck contactor cause loss of AC_1

Furthermore, a stuck closed contactor in combination with a generator fault (Figure 17) or in combination with a short circuit (Figure 19) can cause a loss of voltage on the AC_1 busbar, the probability of which has been computed for a duration of $t = 1$ Fh (flight hour) as $4.0 \cdot 10^{-4}$.

The failure of AC_1 is dominated by faults of the connected cables. If necessary, this situation can be improved by introducing contactors with current limitation between the cables and the busbar, which

prevent the propagation of the effects of short circuits.

For most of the single component faults, this would avoid the effect of losing voltage on the AC_1 busbar. Since there is multiple redundancy in terms of voltage sources, a decrease of the probability of loss of voltage on AC_1 is then limited only by possible own defects of the busbar.

1st order minimal cut sets:

1-1	Feeder1	O/C	0.0002
1-2	AEssFeed1	S/C	0.0001
1-3	Feeder1	S/C	0.0001
1-4	AC_1	S/C	2·10 ⁻⁷
1-5	AC_1	O/C	1·10 ⁻⁷

2nd order minimal cut sets:

2-1	G1	loss	Feeder2	O/C	8·10 ⁻⁹
2-2	G1	loss	Feeder2	S/C	4·10 ⁻⁹
2-3	GC1	O/C	Feeder2	O/C	2·10 ⁻⁹
2-4	Engine1	loss	Feeder2	O/C	2·10 ⁻⁹
2-5	GC1	O/C	Feeder2	S/C	1·10 ⁻⁹
2-6	Engine1	loss	Feeder2	S/C	1·10 ⁻⁹
2-7	G1	loss	ACTC1	O/C	4·10 ⁻¹⁰
2-8	GC1	O/C	ACTC1	O/C	1·10 ⁻¹⁰
2-9	Engine1	loss	ACTC1	O/C	1·10 ⁻¹⁰
2-10	AEssC1	s.c.	AEssFeed3	S/C	1·10 ⁻¹¹
2-11	ACTC1	s.c.	Feeder2	S/C	1·10 ⁻¹¹
2-12	G1	loss	AC_2	S/C	8·10 ⁻¹²
2-13	G1	loss	AC_2	O/C	4·10 ⁻¹²
2-14	G1	loss	GC1	s.c.	4·10 ⁻¹²
2-15	GC1	O/C	AC_2	S/C	2·10 ⁻¹²
2-16	Engine1	loss	AC_2	S/C	2·10 ⁻¹²
2-17	GC1	O/C	AC_2	O/C	1·10 ⁻¹²
2-18	Engine1	loss	AC_2	O/C	1·10 ⁻¹²
2-19	Engine1	loss	GC1	s.c.	1·10 ⁻¹²
2-20	AC_ESS	S/C	AEssC1	s.c.	2·10 ⁻¹⁴
2-21	AC_2	S/C	ACTC1	s.c.	2·10 ⁻¹⁴

Table 1: List of minimal cut sets sorted by probability, for loss of voltage on AC_1 busbar, t = 1 Fh

3.3 Reliability Analysis Example Result

This section shows the result of a dispatch reliability analysis conducted for the introduced electric power system, see Figure 5. The following set of $n_{MEL} = 6$ allowed component deficiencies (MEL-items) is assumed: $C_{MEL} = \{G1, G2, G3, G4, (APU \& AG1), (APU \& AG2)\}$ i.e. six generators two of which in combination with the auxiliary power unit. For $k = 6$ required intact components, i.e. no allowed deficiencies, the analysis determines one minimal path set $MP_1 = C_{System}$ which includes all system components. With given component failure rates λ_i (not listed due to extensiveness) and a duration of $t = 200$ Fh for 10 consecutive days of flying without maintenance, a dispatch reliability of 0.869 is computed.

For $k = 5$, i.e. one allowed deficiency, 6 minimal path sets are generated

$$MP_1 = C_{System} \setminus C_{MEL} \cup \{G1, G2, G3, G4, (APU\&AG1)\}$$

$$MP_2 = C_{System} \setminus C_{MEL} \cup \{G1, G2, G3, G4, (APU\&AG2)\}$$

...

$$MP_6 = C_{System} \setminus C_{MEL} \cup \{G2, G3, G4, (APU\&AG1), (APU\&AG2)\}$$

The dispatch reliability is 0.911 for $t = 200$ Fh.

For $k = 4$, i.e. two allowed deficiencies, 15 minimal path sets are compiled

$$MP_1 = C_{System} \setminus C_{MEL} \cup \{G1, G2, G3, G4\}$$

$$MP_2 = C_{System} \setminus C_{MEL} \cup \{G1, G2, G3, (APU\&AG1)\}$$

...

$$MP_{15} = C_{System} \setminus C_{MEL} \cup \{G3, G4, (APU\&AG1), (APU\&AG2)\}$$

leading to a dispatch reliability of 0.929 for 200 Fh.

As obvious, allowing system deficiencies for dispatch improves the reliability. This is however limited by the failure probabilities of components that are always required to be intact for dispatch. Also, the degraded system must have sufficient safety margin.

4 Conclusion

This paper outlined the capabilities of the Modelica based modelling and analysis tool ENADOT regarding concept design and optimisation of aircraft on-board electric power systems, which have recently gained in relevance, installed power and criticality.

In addition to means for the dimensioning of electric network components regarding power and weight, the system safety and operational reliability can be evaluated in terms of an automated minimal path sets and minimal cut sets analysis.

Future work will be oriented to a transfer of the analysis methods to other physical domains.

Acknowledgements

The research leading to these results has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) for the Clean Sky [3] Joint Technology Initiative under grant agreement no. CSJU-GAM-SGO-2008-001. The author wishes to thank the electric engineering department of Airbus-France for the kind support and company.

References

- [1] Bineid M, Fielding J P: *Development of an aircraft systems dispatch reliability design methodology*. The Aeronautical Journal, pp. 345-352, June 2006.
- [2] Birolini A: *Reliability Engineering – Theory and Practice*. Fifth Edition, Springer Verlag Berlin, 2007.
- [3] CleanSky project, <http://www.cleansky.eu>
- [4] Heidtmann K D: *Smaller Sums of Disjoint Products by Subproduct Inversion*. IEEE Transactions on Reliability, Vol. 38, No. 3, pp. 305-311, August 1989.
- [5] Moir I, Seabridge A: *Aircraft Systems - Mechanical, electrical and avionics subsystems integration*. John Wiley & Sons Ltd, 2008.

Error-free Control Programs by means of Graphical Program Design, Simulation-based Verification and Automatic Code Generation

Stephan Seidel Ulrich Donath
Fraunhofer Institute for Integrated Circuits
Design Automation Division
Zeunerstrasse 38
01069 Dresden, Germany

Stephan.Seidel@eas.iis.fraunhofer.de

Ulrich.Donath@eas.iis.fraunhofer.de

Abstract

Currently the formalisation in the process of creating automation control programs starts with the programming of the real-time controller. But inconsistencies in the requirements definition and misinterpretations will lead to errors in the program which have to be resolved through expensive software-in-the-loop and field tests. This paper introduces a holistic approach for the formalisation of the control design already in the design phase. It also illustrates the design flow for the model-based creation of error-free control programs. Created by means of graphical editors the system definition, which includes the control algorithm, is transferred into Modelica code and thus the executable system model is used for the simulation-based verification. The simulation results are compared to the requirements. Once these are fulfilled and no further errors found, program code for the real-time controller is generated automatically. In this paper Structured Text for programmable logic controllers (PLCs) according to IEC 61131 is generated. In final software-in-the-loop tests the real-time capabilities of the control program are validated.

Keywords: Graphical program design; Modelica; IEC 61131; Structured Text; Software in the Loop

1 Introduction

During traditional control program development it is often necessary to find erroneous sections in the software and fix such code modules by means of extensive software-in-the-loop simulations. The errors are often caused by wrong interactions between components of the control program which are a re-

sult of inconsistencies in the initial project definition [1]. Such inconsistencies could have been found by an intensive and paper-dominated reviewing process which is often shortened in order to save time. As an alternative the formalisation should no longer be carried out at the coding stage but already at the design stage of the software project.

The formalisation of the design consists of the creation of models which contain not only the structure of the system but also the definition of the functionality. In case the models are executable in a simulation software the simulation-based testing of single components as well as the overall system is feasible. Simulation results will be compared to the project's requirements definition. As soon as differences occur they have to be solved by an iterative adjustment of the models or a more precise process definition. When the overall models functionality complies with all requests from the definition an automatic code generation is executed during which code for the target controller is generated. For the final verification of the generated code software-in-the-loop tests are carried out. In these simulation-based tests the real control program is coupled with machine and operator model components which provide request and response signals.

The control program development which is described in the following sections is based on a case filling machine. This machine's model can be subdivided into the machine model, the control model, and the operator model [2].

The machine model is composed of models of all the relevant subsystems such as electrical, mechanical, and hydraulic systems. All of these models are coded in Modelica [3] and are available as objects in libraries. They are instantiated in the model view and then connected with each other in order to reproduce

the structure and composition in the real world. The machine model acts in its entirety as a model of the physical environment with which the controls interact.

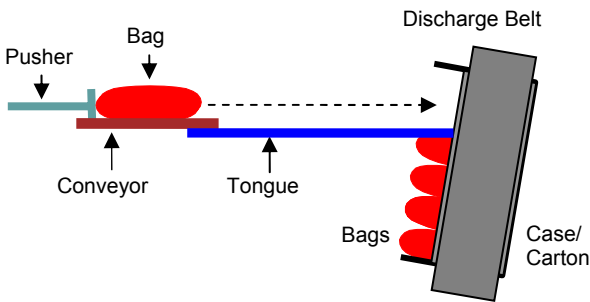


Figure 1: Schematic of the case filling machine

The control model describes the required control operations for the desired course of machine actions. In simple cases the control operations may be linear assignment sequences, but in complex cases they are algorithms which represent state machines. For the graphical description of state machines an UML-profile [4][5] is utilized. These statecharts include simple states, sequential composite states, and pseudostates (initial state, junction, shallow history). The transitions of states are triggered by signal triggers, change triggers, and time triggers and can also be labelled with guards and priority numbers. Priority numbers of transitions are an addition to UML in order to define an evaluation sequence. Activities, which are coded in Modelica, can be assigned to entries, exits, and transitions of states. They are also edited with the graphical editor while creating the statechart. The whole statechart will be automatically translated into Modelica code.

The operator model contains the operating instructions of the system which are derived from the production schedule. They form in general a command sequence and are also given as Modelica code in order to enable an overall system simulation with SimulationX [6].

Each step during the development of the control program

- Graphical program design
- Test and verification at Modelica level
- IEC 61131 code generation [7]
- Software-in-the-loop tests

is illustrated by using the example of the feeding device of the case filling machine.

This machine conducts the final packaging of bags containing products into shipping cartons. The bags are transported on a conveyor from the production process to the feeding device shown in Figure 1.

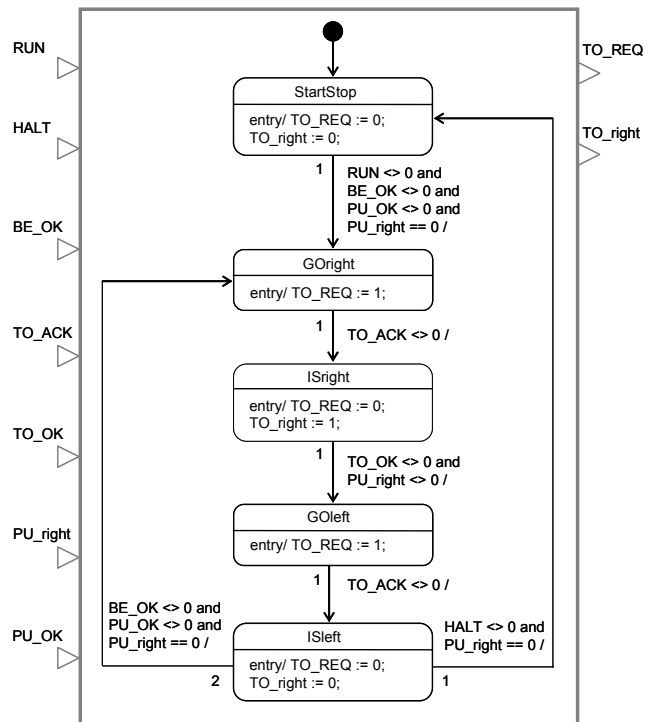


Figure 3: Statechart of the tongue controller

They are placed by the conveyor in front of a pusher on a tongue. The carton is positioned by a belt directly in front of the pusher. As soon as the carton has arrived at this position the tongue is extended and

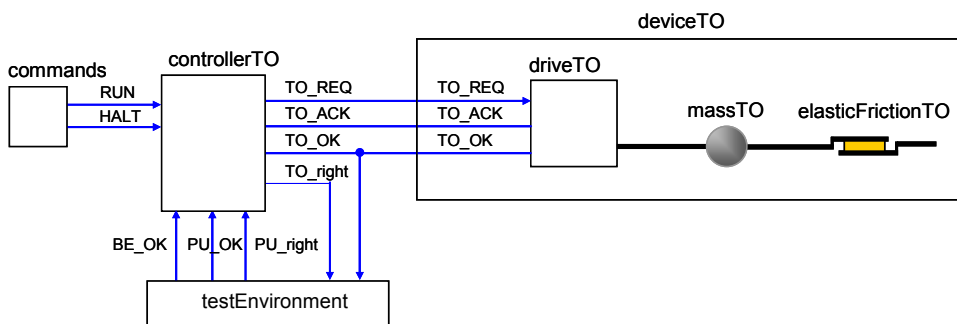


Figure 2: Structure of the tongue's model

the bag is pushed by the pusher over the tongue into the carton. Thereafter the tongue and pusher are retracted and the carton is repositioned to the next filling position.

This paper is organised as follows. Section 2 introduces the graphical design approach. Thereafter section 3 demonstrates the simulation-based verification of the system. The code generation for Modelica and IEC 61131 is detailed in section 4 whereas section 5 illustrates the software-in-the-loop simulation and its benefits. The paper closes with a conclusion in section 6.

2 Graphical program design

At first the models for *tongue (TO)*, *pusher (PU)* and *discharge belt (BE)* are designed. Each component is separated into its control model and machine model.

The controls of tongue and pusher are implemented as state machines (Figure 3). In the entry activities of the states the signal REQ is set or reset. The drive will move thereupon the tongue or pusher back and forth. Transitions between states contain Boolean conditions which act as change triggers. Such triggers fire when the corresponding condition is evaluated to true. In this example these triggers are the feedback from the drive ACK and logical combinations of OK, RUN, and HALT with status signals from the cooperating devices.

The model of the component discharge belt describes the up and down movement of belt and carton. Apart from drive, mass, and friction, the weight is also registered which accumulates as the carton is filled with bags. The interface (Figure 4) includes the signals SPEED, EJECT, ZERO, and POS. Input signal SPEED is the nominal value for the carton's velocity. After the carton is filled, the setting of input signal EJECT starts the ejection of the full carton and thereafter the insertion of an empty one. Output value ZERO indicates the arrival of the carton at the

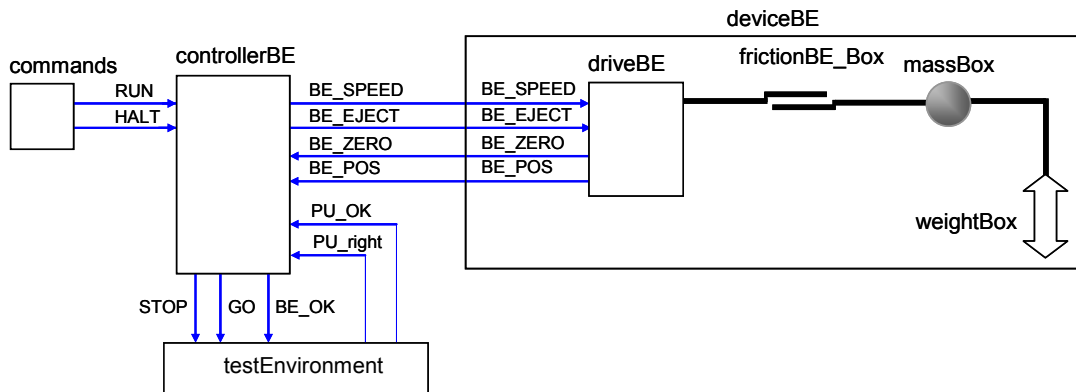


Figure 4: Structure of the discharge belt's model

The machine models of tongue and pusher have the same internal structure consisting of drive mechanism, mass, and friction and show an equal device interface (Figure 2). The setting of the component's input signal REQ (request) will start the positioning of pusher or tongue while the direction of the movement is alternated. The component's output signal ACK (acknowledgement) affirms the movement request whereas OK indicates the request's completion.

The interface of the control is modelled inversely to the device interface. Additional inputs are RUN and HALT to start and stop the operations (Figure 2). Furthermore signals are defined which contain the status of the cooperating devices or send the own status to other devices.

zero position whereas POS indicates its current position. The component's interface is completed with signals RUN and HALT and diverse status signals.

As shown before the control operation is described by a statechart (Figure 5) which groups the belt's movement into a starting phase (states: RunUp, SlowToMin, SlowToNull) and a cyclic positioning phase (RunMax, RunMin, SlowToNull) for filling the carton. In the entry activities of the states the belt's velocity SPEED is set and the target position is calculated. Transition triggers are the ZERO signal as well as the current belt position POS in relation to the target position. Right before the start of the movement the pusher's activity is checked to avoid damage caused by a collision between pusher and carton. The cyclic repositioning of the belt is stopped as soon as the carton is full and the carton is ejected.

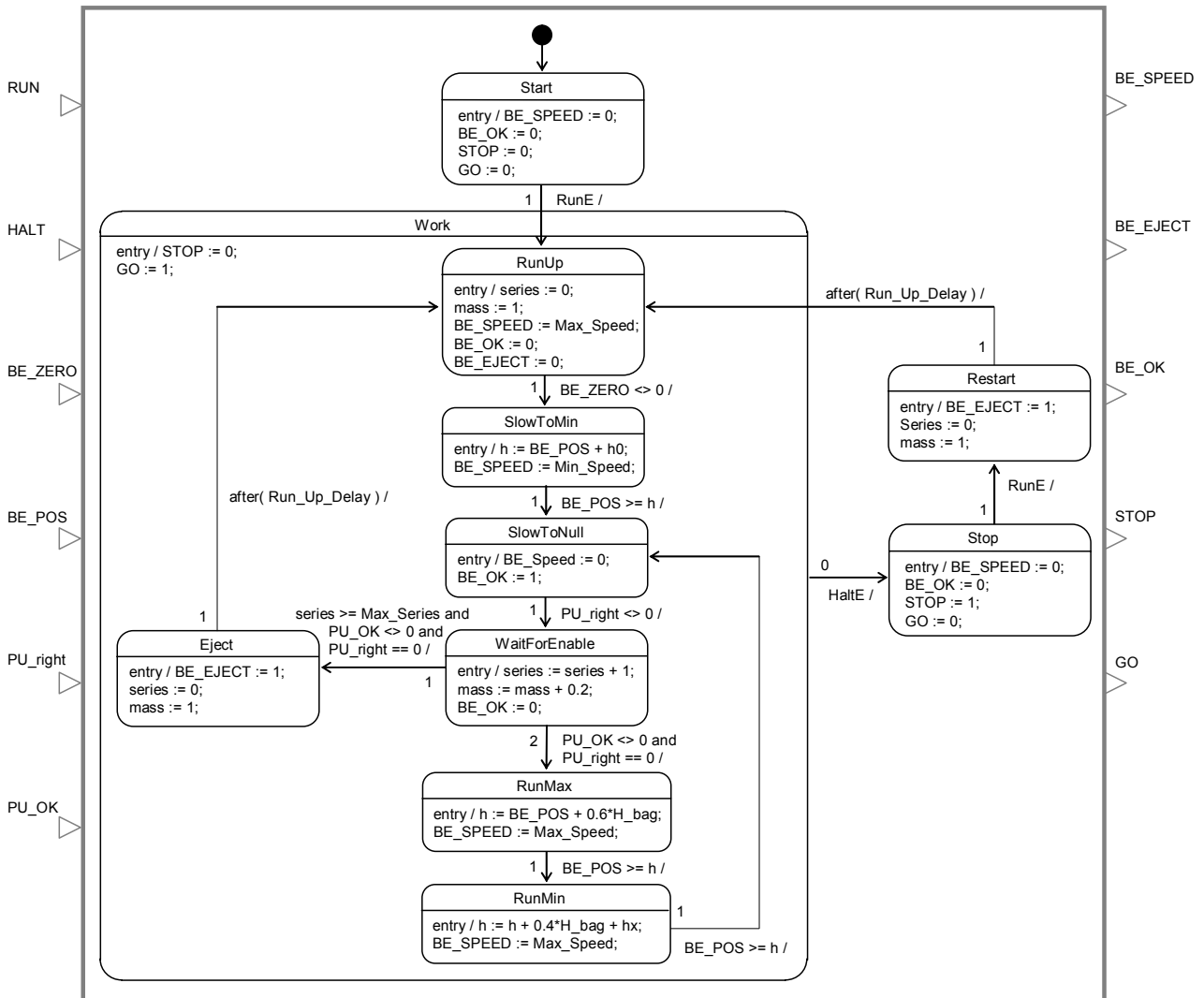


Figure 5: Statechart of the discharge belt controller

After the design of each component (pusher, tongue and discharge belt) they are instantiated in the model view and connected with each other (Figure 6). The belt controller takes over the master function by activating the GO and STOP signals as well as the OK signal after successful positioning for the subordinate controllers of pusher and tongue.

3 Test and verification at Modelica level

Models of the devices are built from physical elements such as mass and friction and from signal blocks which are also part of the library and possess a representation in Modelica. The control components are each translated automatically into a Modelica *model* with corresponding type, variable, parameter, and signal declarations as well as an *algorithm*

section. The structure of this code is illustrated in section 4.1. Figure 6 shows the overall Modelica model which consists of all aforementioned components and their connections.

At first for the verification of the controls, the simulation results of pusher and tongue are compared to the requirements. In Figure 2 the structure of the tongue’s model is shown. The model of the pusher has an equal structure. This connection of control model and device model can be seen as test bench. A path-time diagram for tongue and pusher in relation to the RUN and HALT commands is shown in Figure 7.

The simulation results are: The tongue is retracted from the carton before the pusher is retracted and thus the bag will remain in the carton. The RUN command starts the operation whereas the HALT command stops tongue and pusher after they have reached their initial positions.

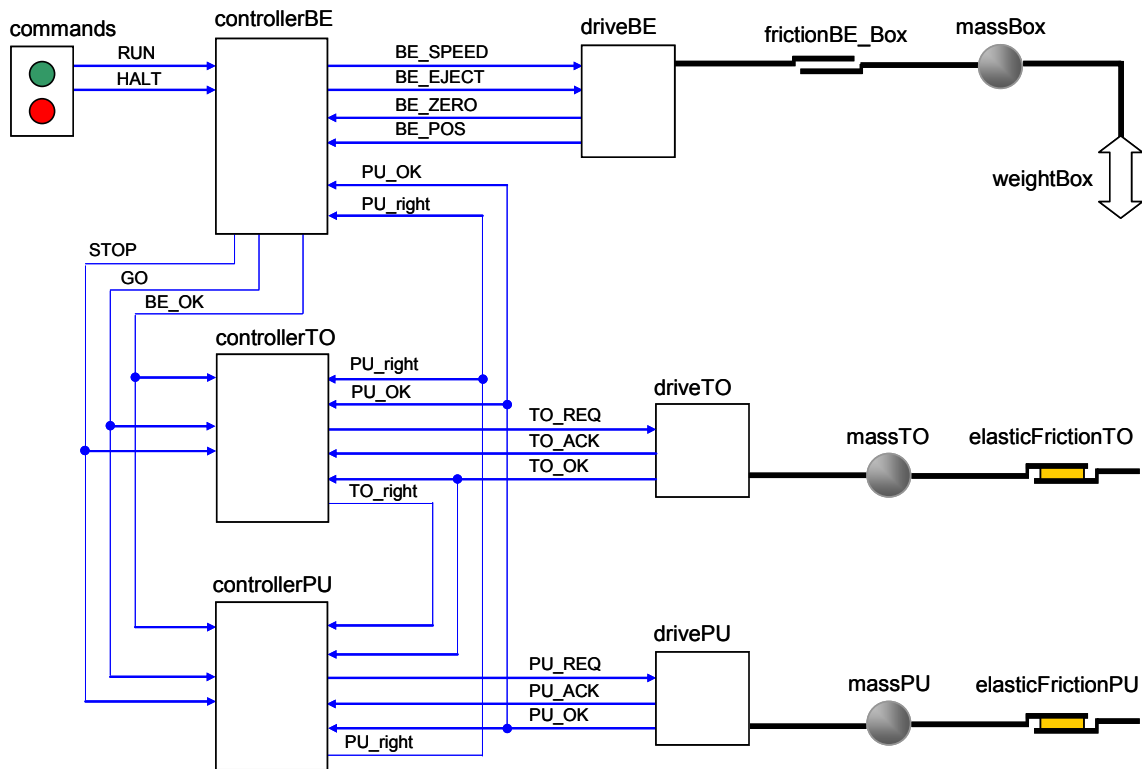


Figure 6: Structure of the case filling machine's overall system model

Secondly, the discharge belt's control is tested in its own test bench, shown in Figure 4. Thirdly, the models of tongue, pusher, and discharge belt are combined into an overall model of the case filling machine as depicted in Figure 6. The simulation results, shown in Figure 8, illustrate the belt position in relation to the movement of tongue and pusher as well as the value of the belt speed. The requirement that the belt moves only in the case that tongue and pusher are in the initial positions has been fulfilled. The HALT and RUN signals along with a filled carton result in a coordinated stop and restart of the course of events.

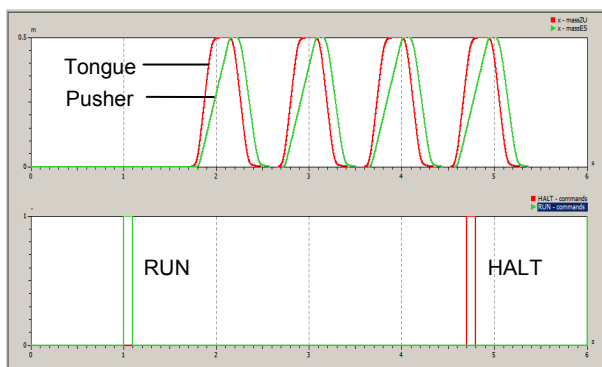


Figure 7: Path-time diagram of pusher and tongue

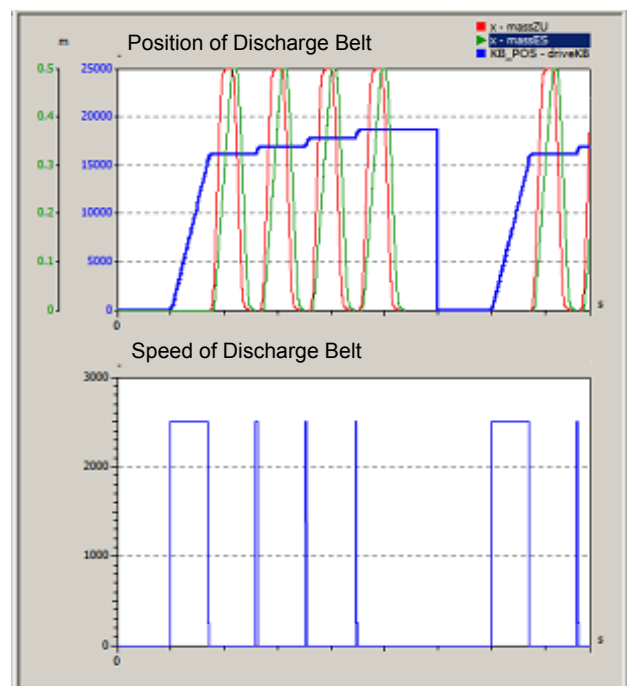


Figure 8: Simulation results of the overall model

4 Modelica and IEC 61131 code generation

4.1 Structure of the Modelica control program

The statecharts of the controllers of pusher, tongue, and belt are translated separately into Modelica code without any manual assistance. Figure 9 highlights the general code structure of these controllers.

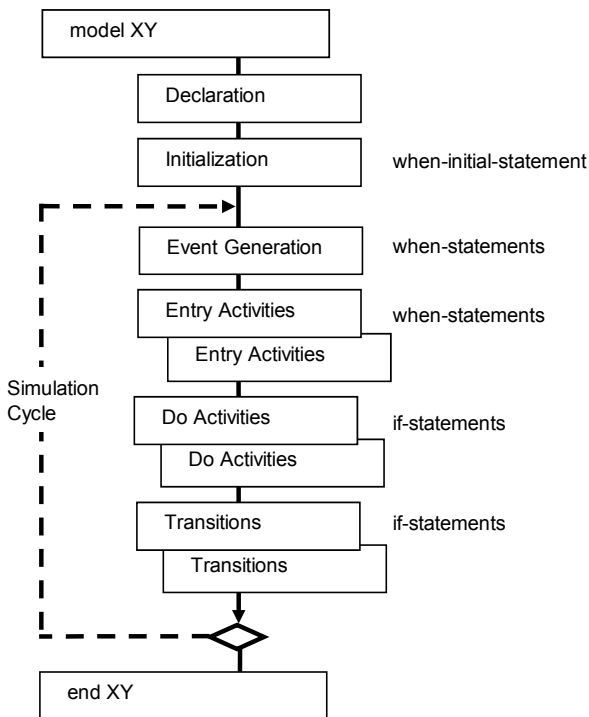


Figure 9: Structure of the statechart's Modelica code

The algorithm section contains the following blocks:

- Event generation:
when-statements with relations of signals or times in order to toggle flags
- Entry activities:
when-elsewhen-statements to check discrete state variables and enclosed *signal assignments*
- Do activities:
if-elseif-statements to check discrete state variables and enclosed *simple signal assignments*
- Transitions:
if-elseif-statements to check discrete state variables and to evaluate Boolean transition conditions and enclosed state assignments to the state variables and *signal assignments*.

The *Entry Activity Blocks*, *Do Activity Blocks* and *Transition Blocks* are created according to the hierarchy of the corresponding statechart. There is no hierarchy existent in the tongue's and pusher's controls (Figure 3). Hence they are translated into one *Entry Activity Block* and one *Transition Block* respectively. The belt's controls show a two level hierarchy which comprises the composite state *Work* and additional simple states (Figure 5). After code generation the two-level hierarchy is represented by two *Entry Activity Blocks* and two *Transition Blocks*.

4.2 Structure of the PLC program

The Modelica code's semantics are the standard for the code of the target controller. In this paper Structured Text (ST) according to IEC 61131 [7] is chosen as target code language. ST is analogue to Pascal and can be used to write programs for programmable logic controllers (PLCs).

The code sections in ST are similar to the statechart's Modelica code and can be separated into the *Event Generation*, *Entry Activities*, *Do Activities*, and *Transition* sections. In addition, the internal program structure requests a separation of declaration and executable functions.

I/O signals and global variables as well as the names of all implemented functions are noted in the symbol table. This table is created automatically when the target controller's code is generated [8]. Local data is defined and stored in a data block which is also part of the generated ST code.

The function *FC Event Generation (EG)* scans for signal events and sets or resets the corresponding flags. Because there is no *when*-statement in ST, edge detection is achieved by using additional flag variables and *if*-statements. An ST code example is shown below:

```

// Event Generation Block
IF NOT("DBSC1".trigEventFlag) AND ("in1">0)
THEN
    "DBSC1".trigEvent := TRUE; //SignalEvent
ELSE
    "DBSC1".trigEvent := FALSE; //SignalEvent
END_IF;
IF ("in1">0) THEN
    "DBSC1".trigEventFlag := TRUE; //EventFlag
ELSE
    "DBSC1".trigEventFlag := FALSE; //EventFlag
END_IF;
    
```

All entry activities of the states are contained in the function *FC Entry Activities (EA)*. A *case*-statement detects the active state according to the state variable. Thereafter the one-time activation of the entry activity is ensured by the implementation of a entry activity flag within an *if*-statement:

```
// Entry Activity Block of Main_State
CASE "DBSC1".S_1 OF //StateVariable
3: IF ("DBSC1".A_1 <> 3) THEN
    out1:=false; //EntryActivity
    "DBSC1".A_1 := 3; //EntryActivityFlag
END_IF;
```

Do activities in are also implemented in the FC EA and noted similarly but do not contain an *if*-statement with an entry flag. Hence do activities are executed in contrast to entry activities in every PLC cycle.

The function *FC Transition (TR)* is comprised of transition triggers, exit activities and transition activities. As illustrated before the active state is detected in a *case*-statement. In each branch an *if*-statement exists which evaluates the corresponding transition trigger and holds the exit- and transition-activity. In case the transition's trigger condition is true, the exit- and transition-activity are executed and the state variable is set to the subsequent state:

```
// Transition Block of Main_State
CASE "DBSC1".S_1 OF
3: IF ("trigger" > 0) THEN
    out1:=TRUE; //ExitActivity
    out2:=FALSE; //Trans.Activity
    "DBSC1".S_1:=4; //State-Variable
END_IF;
```

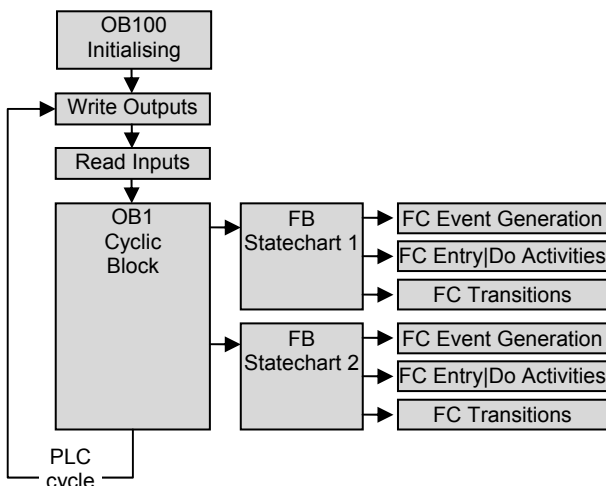


Figure 10: PLC program cycle

All aforementioned functions are called in the function block *FB Statechart* in the following sequence: *EG*, *EA* and *TR*. Timers which are used in the statecharts time triggers are also administrated in this function block. *FB Statechart* is called in organisation block *OB1*, which is the PLC's cyclic main program and is executed right after the update of the input register. The program of *OB1* contains the calls of functions and function blocks which implement the control program functionality. After the execution of each function, the PLC jumps back into *OB1* and when *OB1*'s end is reached the current cycle ends with the update of the outputs. Figure 10 illustrates the PLC cycle and the call sequence of the control program's functions.

At startup of the PLC the initialisation block *OB100* is executed which resets and initialises the state machine's internal variables. Organisation blocks *OB1* and *OB100* as well as function block *FB Statechart* are also part of the automatically generated ST program code. Hence a complete PLC program is generated from the controller's statechart.

After code generation two ASCII-files are available: one file containing the symbol table and one file containing the program code. Either file can be imported into the PLC engineering software Step7 by Siemens [9].

4.3 Compilation of the PLC program from a Structured Text source

The aforementioned transformation of a statechart into Structured Text indicates the similarities to Modelica, but there are additional requirements that need to be fulfilled. The main requirement which is caused by the PLC program's organisation into functions, function blocks, data blocks etc. says that within the ST source code called functions are stated in front of the functions that call them [10].

Therefore the order of the functions in the ST source code has to be as follows:

1. Data block *DB* with internal variables
2. Functions *FC Event Generation*, *FC Entry and Do Activities*, *FC Transitions*
3. Function block *FB Statechart*
4. Organisation block *OB1 Cyclic Block*
5. Organisation block *OB100 Initialising*.

The SCL batch compiler [10] of Step7 transforms the ST source code into MC7 code, which is executed by the PLC, and carries out a lexical, syntactical and semantic analysis and thereafter generates automatically all defined functions and blocks. They are stored in the design environment and can be loaded into as well as executed and tested on the PLC.

In case the automation systems controls are distributed over several statecharts, as is the case with the case filling machine described in section 2, then one set of source files (symbol table and ST code) is generated for each statechart. An add-on tool was implemented which merges all source files into one file set. The tool scans all symbol tables, eliminates double entries and copies all other entries into one table. The ST sources are also combined into one source. Functions and data block of the statecharts remain unchanged only their numbering is adjusted. The main task is to merge all organisation blocks (OB1's, OB100's) into one block of each type as there is only one instance of each type allowed in the PLC. At this point it is very important to consider the call sequence of the state machines on the PLC in order to avoid unwanted effects such as racing conditions caused by the serialisation of the PLC code [12].

5 Software-in-the-Loop tests

The simulation of the automation system can now be taken to a new level which will be illustrated in the following by using the belt controller as an example. After the PLC program has been generated, compiled and loaded a co-simulation of machine model (Modelica simulation) and PLC program (software PLC) can be conducted. The simulation tool PLCSIM [11], which is part of the Step7 PLC engineering system, is applied as software PLC. PLCSIM runs as a separate process on any Windows PC and enables the execution of the PLC program analogue to a hardware PLC. As a simulator PLCSIM offers no interface to hardware I/Os communicating with the field and thus can only be employed for simulation purposes.

In order to perform a co-simulation the inputs and outputs of the PLC have to be assigned to the corresponding signals of the machine model and cyclically updated in both directions.

The Modelica statechart model in the simulator has to be substituted with a coupling element that has exactly the same input and output signals as the statechart block. There are coupling elements available in the system simulator but these do not provide a suitable interface for a connection with the PLC. In addition the PLC's timing needs to be adapted to the system simulators timing or in other words, the PLC can no longer run in real-time but needs to run synchronous to the simulators simulation time.

Hence a coupling tool was designed and implemented which enables the coupling of PLC and system simulator. The tool provides interfaces in either

direction. A TCP-socket connection is used for communication with the system simulator whereas communication with the PLC is achieved via a COM-object. The coupling tool, also called Backplane, is configured and then started by the user.

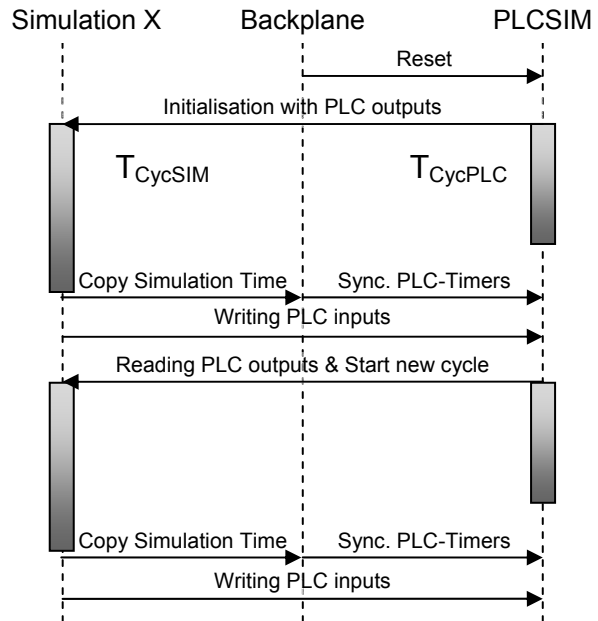


Figure 11: Simulator Synchronisation

Apart from configuration data such as IP-address and port, also the assignment of PLC I/O signals to the system simulator signals is defined.

In addition all PLC timers that are used somewhere in the PLC program have to be noted, as they will be synchronised by the Backplane. The Modelica simulation for operator and machine model is running in simulation time whereas the PLC is running in real-time. Therefore the synchronisation of time is indispensable and so the PLC's timers are synchronised at the beginning of each PLC cycle. Hence the progression of the timers is no longer related to the real-time, but depends only on the simulation time.

After the start of the simulation the Backplane takes over the control of the Modelica simulator and the PLC simulator and performs the data exchange between them. As show in Figure 11 in both simulators the same fixed time interval is simulated and thereafter the simulation is halted. The real-time duration T_{CycSIM} and T_{CycPLC} of the simulation cycles may differ. The Backplane scans the sensor signals of the machine model and writes them to the PLC's input signals. Then the PLC's output signals are read and transferred to the machine model's actuator signals before a new simulation cycle starts. This mode of operation is shown in Figure 11.

After the simulation run the results from the co-simulation are compared to the results of the Modeli-

ca-only simulation. A path-time diagram (discharge belt's position) of these simulation runs is shown in Figure 12. The waveforms are not identical as there is a growing time offset between characteristic points and also a spatial offset. Both effects can be traced back to the PLC's cycle time which is not present at the Modelica model level. The state machine of the controller model responds immediately to a change of the input signals. As soon as an input signal from the machine or operator model triggers a transition the simulation is halted. The Modelica code of the statechart block will then calculate the subsequent state and execute all exit activities, transition activities, and entry activities. This procedure is repeated until a stable state configuration is found, i.e. no further transition can fire. Therefore state transitions consume no simulation time and activities are instantly visible. The simulation is then started again and will run until the next event occurs. This behaviour is caused by the simulator's model of computation.

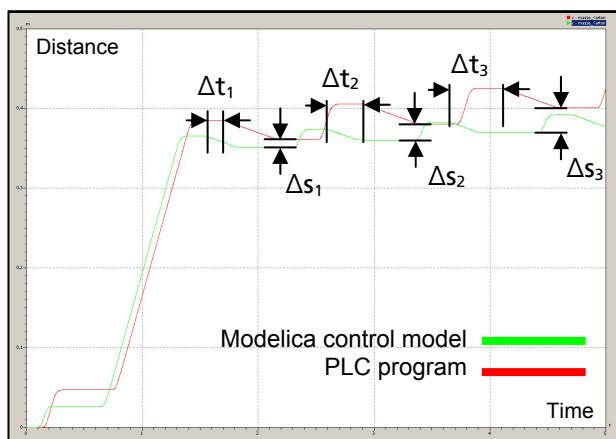


Figure 12: Path-time diagram of belt movement controlled by Modelica control model and PLC program

The PLC processes its control program after a different Execution Model [12]. As already discussed in section 4.2 the controller is executing the control program in a cyclic manner. Therefore the calculation of a new output signal vector requires one or more PLC cycles depending on the call sequence of the functions that were generated from the UML statechart. Typical cycle times for industrial controllers are 5 to 50 ms. The call sequence of the functions *EG*, *EA*, *TR* is also a cause for serialisation effects such as racing conditions. In case more than one state machine is used for controlling the system the sequence of their execution is also important. Such effects are not covered by the Modelica control model and can hence not easily be simulated. It is therefore necessary to extend the control model with

an execution model of the real-time controller. At present the authors implement an execution model for PLCs at model level. The execution model is also designed as statechart and thereafter translated into Modelica code. First results of this approach are discussed in [12].

The main cause for the different behaviour of Modelica control model and real-time controller is the cycle time. For time-critical functions such as fast positioning operations a large cycle time can be unfavourable and result in the example of the case filling system in delays and spatial offsets when positioning the discharge belt as depicted in Figure 12.

The Backplane provides the option to force the cycle time. Therefore simulation runs with different cycle time can be simulated and the effects on the results analysed. This data can then be employed to find an upper limit of the cycle time which guarantees the correct and exact function of the system. The determined value is a criterion for the selection of a PLC.

The generated PLC program is tested against predefined test cases and the results are scanned for errors or deviations from the control models results. All test cases should encompass an extensive amount of input signal combinations and not only valid but also erroneous signal combinations. In case the control programs behaviour and function was correct the PLC program can be transferred to the real system without further adaptations.

6 Conclusion

In this paper the model-based design and simulation-based verification of an automation system as illustrated in Figure 13 was presented. Following the example of an industrial case filling machine the modelling of its control with statecharts was demonstrated. These statecharts were transformed automatically into Modelica code and then executed in test benches. For verifying the control algorithms the Modelica control blocks were stimulated with test signals and their behaviour was improved until the function corresponded to the defined requirements. Thereafter the Modelica blocks of control model, machine model and operator model were combined to an overall system model and a system simulation was executed by which the system's function was verified and important performance indicators were established. Once the behaviour and performance of the system was correct the IEC 61131 Structured Text code for the real-time controller, a PLC, was generated. The PLC's control program was then tested and validated

as software-in-the-loop against the machine and operator model. This procedure showed a satisfying behavioural consistency between Modelica control model and real-time control program and only minor deviations were detected. These effects can be traced back to the PLC's cycle time. In order to overcome such deviations the authors are working on the implementation of an execution model for real-time controllers in Modelica.

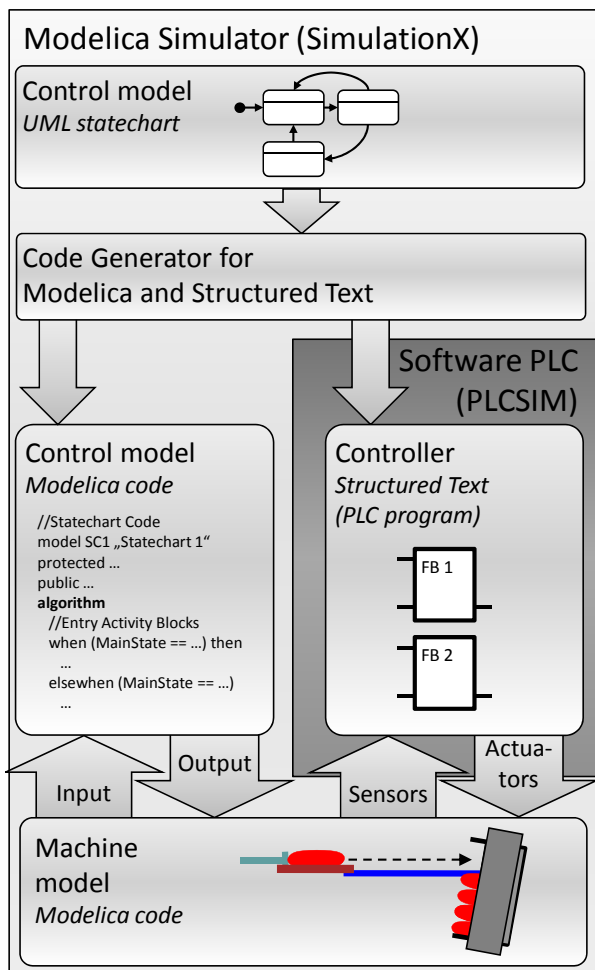


Figure 13: Design flow for modelling and verification of machine controls

The approach discussed in this paper removes the need for the error-prone manual coding of the PLC program and saves a huge amount of time by minimising the field and start-up tests through simulation-based verification of the control algorithms. The design-flow is faster and much more efficient than the current state-of-the-practice procedures and provides an easy way to error-free control programs through graphical and model-based design, simulation-based verification, and automatic code generation. The transformation steps in this chain are carried out automatically so that human efforts can be focused on the design of the controls and its evaluation.

References

- [1] Schwabe, S.: Modellbasierter Systems-Engineering-Prozess. ECONOMIC ENGINEERING 3/2009, S. 58-59
http://www.berner-mattner.com/.../BernerMattner_Fachartikel_ModellbasSystemsEngProzess.pdf, visited on: 20.01.2011
- [2] Haufe, J., Donath, U., Lantzsch, G.: Modellbasierter Entwurf von Steuerungen in der Automatisierungstechnik. Dresdner Arbeitstagung Schaltungs- und Systementwurf (DASS), Dresden, March 2009
- [3] <https://www.modelica.org/documents/ModelicaSpec30.pdf>, visited on: 20.01.2011
- [4] OMG Unified Modeling Language Superstructure V2.2
- [5] Donath, U.; Haufe, J.; Blochwitz, T.; Neidhold, T.: A new approach for modeling and verification of discrete control components within a Modelica environment. Proceedings of the 6th Modelica Conference, Bielefeld, March 2008, p. 269-276
- [6] <http://www.simulationx.com>, visited on: 20.01.2011
- [7] Int. Electrotechnical Commission: IEC Standard 61131-3: Programmable controllers - Part 3, 1993
- [8] Lindner L.: Rapid Control Prototyping by Transformation of Hierarchical State Machine Control Models into IEC 61131 PLC Code. Diploma thesis, TU Dresden, 2009.
- [9] Siemens AG. Software for SIMATIC controllers.
<http://www.automation.siemens.com/mcms/automation/en/automation-systems/industrial-automation/Pages/Default.aspx>, visited on 20.01.2011
- [10] Siemens AG. S7-SCL V5.3 for S7-300/400 Manual, 2005
- [11] Siemens AG. SIMATIC S7-PLCSIM V5.4 Manual, 2007.
- [12] Seidel S., et al.: Modelling the Real-time Behaviour of Machine Controls Using UML Statecharts. Proceedings of the 15th International IEEE Conference on Emerging Technologies and Factory Automation, Bilbao, September 2010

Enforcing Reliability of Discrete-Time Models in Modelica

Sébastien FURIC

LMS Imagine

7, place des minimes 42300 Roanne

sebastien.furic@lmsintl.com

Abstract

Modelica models involving discrete-time aspects may lead to surprising results due to the way events are currently handled in the language. Indeed, *simultaneity* is interpreted as *synchronism* (see [2] for details) and, as a consequence, two unrelated sources of events may interfere in unexpected ways.

In this paper, we present minimal examples of models that exhibit unexpected or surprising results, then we explain the general causes of such behaviors and propose to introduce the notion of *clock* in the language to solve the issues. In contrast to [1] and [2], we focus here on models resulting from the *composition* of other models: we aim at showing that the current discrete-time theoretical model of Modelica is not robust with respect to model composition. For the final user, it means that it is generally not possible to build reliable models involving discrete-time aspects by simply connecting generic library models: manual adjustments are often required to obtain the expected behavior¹.

Keywords: discrete-time modeling; clock calculus

1 Introduction

Modelica has been designed to primarily solve *continuous-time systems* of differential and algebraic equations. Unfortunately, discrete-time aspects have not been considered with the same level of interest. The result is that essential features of synchronous languages (e.g., Signal, Lustre, Esterel) are not present in Modelica today. Consider for instance the following Modelica model:

```

model M
  Real x, x_dot;
  Integer count;
initial equation
  x = 0;

```

¹ This also begs for a related question, which is: how can we *know* that our models actually require adjustments!

```

  x_dot = 1;
  count = 0;
equation
  x_dot = der(x);
  der(x_dot) = -x;
  when
    { x > 0.5, sin(time) > 0.5 }
  then
    count = pre(count) + 1;
  end when;
end M;

```

According to the Modelica specification, that model is correct, so we can try to simulate it. One may wonder which is the value of `count` at the end of a simulation performed between 0 and 100 seconds for instance. Quite surprisingly, the Modelica specification does not give the answer: any value between 16 and 32 is possible even if — it is the case here — every event can be numerically detected with accuracy so that none is lost due to the limits of time tolerance of the solver². Indeed, the *when clause* that is used to update `count` is activated by two unrelated sources of events (put between curly braces in Modelica syntax) that may accidentally be seen as synchronous during simulation, as explained in [2]. Actually the final value of `count` depends on:

- the “quality” of the translator implementation
- the kind of solver eventually required to solve the final system
- the parameters of the solver, in case a solver is necessary.

In this paper, we aim at explaining the consequences of such a design choice in terms of reliability and reusability of models. The paper is organized as follows: section 2 gives an analysis of the prob-

² The purpose of this paper is not to discuss numerical solver issues, in particular event detection in case of non-trivial continuous-time systems: we will only focus on discrete-time aspects. The introductory example is presented with the hope that it will help readers with physical background to get a feeling of what discrete-time issues are.

lem; section 3 introduces the proposed solution; section 4 shows how to transpose the solution in the context of the Modelica language; section 5 presents an example of application; section 6 gives a conclusion.

2 Analysis of the problem

The problem with the model above comes from the fact that discrete-time aspects are somewhat “approximated” in Modelica's semantics: it is not possible to know for sure, in that model, whether both sources of events corresponding to $x > 0.5$ and $\sin(\text{time}) > 0.5$ are synchronous or not. It is not even possible to know for sure, in case they are seen as synchronous by the simulator at the beginning of a simulation, whether they will remain synchronous until the end or not. Indeed, according to the Modelica specification, events instants are “probed” during simulation (only time associated to their occurrence is retained) so deciding whether two events happening at the same measured time are really synchronous (i.e., have the same cause) or whether it is pure coincidence is impossible. Unsurprisingly, this has unfortunate consequences over the design of event-based models in Modelica. Consider for instance the following purely discrete Modelica models:

```
connector Out = output Boolean;
```

```
model EventSource "Simple event source"
  parameter Real t0, T;
  Out out;
equation
  out = sample(t0, T);
end EventSource;
```

```
connector In = input Boolean;
```

```
model Counter "Simple event counter"
  parameter Integer n;
  In ins[n];
  Integer count;
initial equation
  count = 0;
equation
  when ins then
    count = pre(count) + 1;
  end when;
end Counter;
```

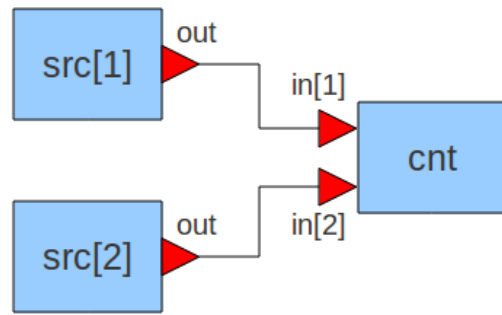


Figure 1: A simple test model

Instances of `EventSource` emit events³ via their unique output port and instances of `Counter` count the number of events received via their input ports. Consider the following model built upon `EventSource` and `Counter` (Figure 1 gives its graphical representation):

```
model TestCounters
  EventSource src[2](t0 = { 0, 3 },
  T = { 1, 2 });
  Counter cnt(n = 2);
equation
  connect(src.out, cnt.ins);
end TestCounters;
```

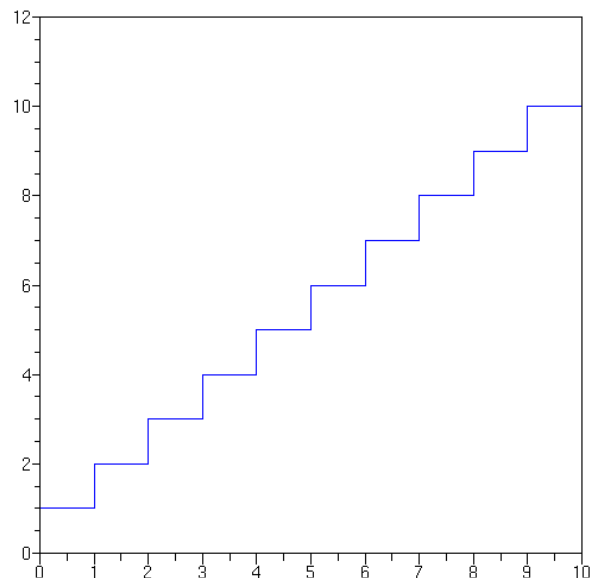


Figure 2: Simulation results of model in Figure 1

Simulation of an instance of `TestCounters` between 0 and 10 seconds gives a rather surprising result (see Figure 2). Indeed, between 0 and 10 seconds the sources emit a total of 11 + 4 events but

³ Boolean values in reality, events are not explicitly emitted.

only 11 of them are “saw” by the instance of Counter. That result is explained by Modelica's way of handling discrete events. Indeed, some events are “lost” because, as explained above, only the *measured time* of events matters in Modelica, so two events happening at the same time cannot be distinguished: the simulator does not know whether they have been emitted by the same source connected to both input ports of the instance of Counter (as in Figure 3) or by two distinct sources (as in Figure 1). One may wonder why is it not possible, by default, to consider every local port of a model like Counter as an independent local source of events: indeed, in the case of TestCounters, it would give the correct answer for count. But consider the following model which graphical representation is given in Figure 3:

```

model TestCounters2
  EventSource src(t0 = 0, T = 1);
  Counter cnt(n = 2);
equation
  connect(src.out, cnt.ins[1]);
  connect(src.out, cnt.ins[2]);
end TestCounters2;
    
```

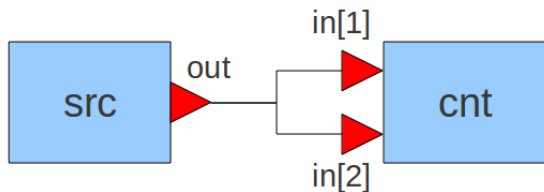


Figure 3: Another simple test model

In any instance of that model, if every port of cnt would be considered as a local source of events then twice the correct number of events would be found since there is only one real source of events (with duplicated outputs). One may notice that Modelica's default behavior would lead to the correct result (by accident, however) in that very special situation.

Going back to the original model, one way to avoid the event loss problem in Modelica would be to associate one “subcounter” per input port and to sum the results into the global counter count, as in:

```

connector In = input Boolean;

model ImprovedCounter
  parameter Integer n;
  In ins[n];
  Integer count;
    
```

```

protected Integer subcount[n];
initial equation
  subcount = zeros(n);
  count = 0;
equation
  for i in 1 : n loop
    when ins[i] then
      subcount[i] =
pre(subcount[i]) + 1;
    end when;
  end for;
  when ins then
    count = sum(subcount);
  end when;
end ImprovedCounter;
    
```

However, this solution is far more space- and time-consuming than the original Counter model (because a number of additional state variables proportional to the number of listened sources has to be declared and the whole sum of subcounters has to be recomputed each time an event is detected on any input port). Also, that new solution still fails to count the correct number of events in case of a configuration like the one in Figure 3. We may even want to consider configurations like the one in Figure 4, where neither Counter nor ImprovedCounter would give the correct answer.

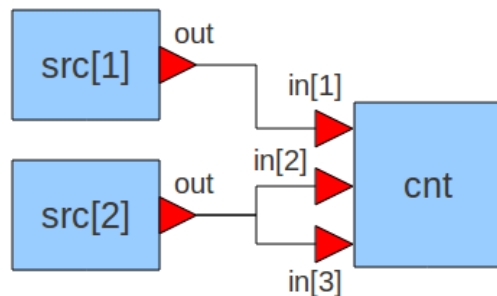


Figure 4: A slightly more complex version of previous test models

A last remark can be made regarding correctness of models. Going back to the first test model, we managed to correct it by providing ImprovedCounter (an *adapted version* of Counter) to circumvent the issue with simultaneous events. It is important to notice that the correction was possible because we *knew* that our original model had problems with respect to event handling. But in real-world situations, where correctness of models is not known *a priori*, Modelica compilers will not be able to detect such errors since, as shown above, the Modelica language itself does not retain the required information.

As a result, users will have to determine by hand whether their models are correct or not. Of course the task is impossible to complete as far as models get too big or contain encrypted parts for instance.

We conclude from those observations that Modelica needs some improvements to enable the definition of reliable models involving discrete events. The following section explains how that can be achieved. The proposal is based on a preliminary work by INRIA and LMS Imagine in the course of the SimPA 2 project ([1], [2]).

3 Proposal to enable the definition of reliable discrete models in Modelica

3.1 Introduction to clocks and signals

The most important feature of synchronous languages that is currently missing in Modelica is *clocks*. In the Signal language ([3]) clocks give *logical instants* at which signals are said to be *present*, i.e. instants at which values of signals are accessible. Signals sharing the same clock are said to be *synchronous* (their values are present at the same logical instant). Clocks give the *domain* of signals, and types (e.g., Boolean, Integer, Real, etc. in Modelica) give their *codomain*. Consider the following Modelica program:

```

model M
  Integer count;
initial equation
  count = 0;
equation
  when sample(0, 1) then
    count = pre(count) + 1;
  end when;
end M;

```

Interpreted in terms of clocks and signals, this program would define the *discrete-time signal count*. One way to see `count` would be as a mapping from events to values (`Event` is the set of all events):

```

count:      Event → Integer
           e0 → 1
           e1 → 2
           e2 → 3
           ...

```

Of course, we would also need to associate a “physical time” with each event, as required by the definition of `sample()`:

```

e0 ↦ 0.0
e1 ↦ 1.0
e2 ↦ 2.0
...

```

It is fundamental to notice that the mapping from events to physical time is not a bijection: two distinct events may be associated with the same physical instant, in which case those events are said to be *simultaneous*. We saw that in Modelica there is no way to tell whether two simultaneous events have the same origin since we only look at physical time. By looking at logical instants, we have a more accurate view of the flow of events: that is the basis of the *synchronous approach* to event handling.

3.2 Why do clocks and signals solve the issues

Let's consider an expression like `sample(t0, T)`. Interpreted in terms of signals and clocks, it would represent a sequence of *fresh events*, each of them mapped to physical instants so that e_k ($k \geq 0$) maps to $t0 + kT$. Consider the following program:

```

when sample(0, 1) then
  count = pre(count) + 1;
end when;

```

We say that the *when clause* above is *activated* at each logical instant yield by the `sample` construct, which defines a clock, and that `count` *inherits* that clock: `count` causally depends on the `sample` construct that is used to activate the equation.

We now introduce the notion of *clock union*, as in:

```

when { c1, c2, ... } then
  count = pre(count) + 1;
end when;

```

{ `c1`, `c2`, ... } represents the union clock of `c1`, `c2`, etc. The set of events emitted by that clock is the union of the set of events emitted by each clock used to compose it. And since in our interpretation events can be distinguished one from each other we can, contrary to Modelica with its current interpretation, compute the union accurately:

- without accidentally forgetting any element (as illustrated in `TestCounters` above)
- without accidentally counting the same element twice (as illustrated in `TestCounters2` above).

Of course, taking clocks into account requires a less naive compilation process than those currently implemented in Modelica compilers. In the next sections we describe the steps required to transform a synchronous language program into efficient compiled code.

3.3 Considerations about the compilation of synchronous programs

Take for instance the model class `TestCounters` defined above. If we instantiate it, we get the following flat Modelica program (where connection equations have been replaced by their contribution to the final system of equations):

```

model FlatTestCounters

  // variables introduced by the
  first event source
  parameter Real src[1].t0,
  src[1].T;
  Boolean src[1].out;

  // variables introduced by the
  second event source
  parameter Real src[2].t0,
  src[2].T;
  Boolean src[2].out;

  // variables introduced by the
  counter
  Boolean cnt.ins[1], cnt.ins[2];
  Integer cnt.count;

initial equation

  // initial equations introduced
  by the counter
  src.count = 0;

equation

  // equations introduced by the
  first event source
  src[1].out = sample(src[1].t0,
  src[1].T);

  // equations introduced by the
  second event source
  src[2].out = sample(src[2].t0,
  src[2].T);

  // equations introduced by the
  counter

```

```

when { cnt.ins[1], cnt.ins[2] }
then
  cnt.count = pre(cnt.count) + 1;
end when;

// expanded connection equations
cnt.ins[1] = src[1].out;
cnt.ins[2] = src[2].out;

end FlatTestCounters;

```

For the sake of conciseness, we will consider a simplification of the previous program⁴ that still contains the essential constructs:

```

model ShortFlatTestCounters
  parameter Real t0[1], T[1],
  t0[2], T[2];
  Boolean c[1], c[2];
  Integer count;
initial equation
  count = 0;
equation
  c[1] = sample(t0[1], T[1]);
  c[2] = sample(t0[2], T[2]);
  when { c[1], c[2] } then
    count = pre(count) + 1;
  end when;
end ShortFlatTestCounters;

```

That program defines two asynchronous event sources since we consider that each sample construct introduces its own sequence of fresh events. It follows that `c[1]` and `c[2]` do not have any event in common and then that the union clock `{ c[1], c[2] }` is irreducible. The *clock calculus* we will propose below will have to reflect those considerations, so that a compiler implementing it will automatically derive canonical representation of clocks, as we currently do by hand in this simple case. The constraints in the above program are finally equivalent to this pseudo-code:

```

c[1] = sample(t0[1], T[1]);
c[2] = sample(t0[2], T[2]);
when c[1] then
  count = pre(count) + 1;
end when;
when c[2] then

```

4 Obtained by removing alias variables and associated equations, and renaming remaining variables

```
count = pre(count) + 1;
end when;
```

Notice that we now have two concurrent equations defining `count` (which is explicitly forbidden in Modelica) but since both when clauses are guaranteed to be activated asynchronously thanks to our interpretation of `sample`'s properties, there is actually no possible conflict⁵.

At this point, an important remark has to be made regarding *determinism*. Indeed, with the semantics proposed in this report, we accept to consider that a connection of two independent event sources is acceptable because we force *interleaving of events* so that two simultaneous events cannot be synchronous if they don't have the same origin. However, the order into which simultaneous events will be treated at runtime is purposely left unspecified. From a control engineer perspective that choice seems rather surprising, especially for a language that could eventually be used to design control systems, where determinism is a fundamental aspect to consider. But from a physical modeling point of view, non-determinism is a natural consequence of the physical nature of the world. Since our aim is to design a physical modeling language it seems reasonable to allow some form of non-determinism to avoid rejecting too many programs, especially those that most users having a physical background will consider correct (and which are, given the limits of physics). Notice however that non-determinism can be detected statically by a compiler implementing our proposal: it just requires a stricter criterion to select correct programs (that may be a compiler option). Compared to Modelica in its current state, we offer a way to control determinism so that it fits control or physical needs. Modelica, on the other hand, currently cannot promise anything regarding determinism for the reasons exposed in previous sections.

3.4 Clock calculus

In this section we introduce the elements of our *clock algebra* and give essential rules that govern clock calculus. We propose the following grammar to describe clock expressions (where e represents a term denoting any signal, and b a term denoting any boolean signal):

```
clock ::=
  never          (empty clock)
| always        (full clock)
| clock(e)      (clock of e)
```

5 Modelica currently has to impose single assignment restrictions precisely because two sources of cannot be statically proven to be asynchronous, as explained before.

```
| false(b)      (instants at which b is false)
| true(b)       (instants at which b is true)
| initial       (initialization clock)
| edges(b)      (instants at which b becomes true)
| sample(t0, T) (sample starting at t0, with period T)
| clock ∨ clock (union of clocks)
| clock ∧ clock (intersection of clocks)
| clock \ clock (difference of clocks)
| c1, c2, ...  (clock variables)
```

Several comments have to be made:

- we introduce the notion of “full clock”, which, in synchronous languages such as Signal, makes no sense (since clocks are discrete). But since here we have to consider continuous-time signals (we want to describe DAE systems among others), we have an implicit maximal clock for any program: the default continuous-time clock, which includes all the instants of the simulation
- the same kind of remark can be made for clock difference: if the first argument is the full clock, then we get the complementary of the second argument as result, which also makes no sense in synchronous languages such as Signal
- **sample()** and **edges()** share the following properties:
 - they are *generative*: they always yield a fresh, pure discrete-time clock
 - two clocks yielded by those constructs are guaranteed to have an empty intersection (which implies for instance that $\text{edges}(e_1) \wedge \text{edges}(e_2) = \text{empty}$ for any e_1, e_2 ⁶)
- **initial** is a special clock that contains only one instant which corresponds to the first simulation instant (i.e., no other instant may happen before this one).

Systems of equations will be described by the following grammar (where $e_i(S_{i_1}, S_{i_2}, \dots)$ denotes an expression involving signals S_{i_1}, S_{i_2}, \dots):

```
system ::=
  null          (system having no constraint)
| system || system (parallel composition)
| system when clock (sampling)
| let c = clock in system (let binding)
| e1(S1,1, S1,2, ...) = e2(S2,1, S2,2, ...) (equation)
```

To illustrate the use of the system description language defined above, let's write the system corresponding to an instance of the model class `TestCounters` defined above. It gives:

6 This is the property we used in section *Considerations about the compilation of synchronous programs* to calculate the union clock.


```

let c[1] = sample(tO[1], T[1]) in
let c[2] = sample(tO[2], T[2]) in
count = 0 when initial ||
count = pre(count) + 1 when c[1] v
c[2]
    
```

We saw in previous section that it was also possible to write it as follow:

```

let c[1] = sample(tO[1], T[1]) in
let c[2] = sample(tO[2], T[2]) in
count = 0 when initial ||
count = pre(count) + 1 when c[1] ||
count = pre(count) + 1 when c[2]
    
```

Notice that clock expressions should be reduced to a *canonical form* so that single assignment rule can be checked statically to avoid current Modelica issues with respect to discrete-time modeling. We then require that a compiler will have to perform *full reduction* of clock expressions at compile time by eliminating local clock variables (i.e., rewriting **let** $v = \dots$ **in** \dots terms) and replacing **clock**(...) terms by their actual value. This reduction will have to be carried out in parallel with the resolution of the assignment problem attached to equations. Indeed, in contrast to synchronous languages such as Signal, we have to deal with the acausal nature of the language: given an equation, it is not possible to tell which signal(s) it defines without performing a *global causality analysis under clock constraints* (we require however *schedulability* of clock constraints). So the algorithm we propose is:

1. The initial contextual clock is **always**
2. Perform partial resolution of the assignment problem for any equation that is not constrained by a when construct, in order to find defined signals, which are added to current context
3. Pick any when construct which activation clock either depends on past (i.e., “pre signals”) or on signals that belong to current context; make that activation clock the current contextual clock and go to step 2

If some subsystems remain unselected at the end of the algorithm, the whole system is:

- not schedulable, if any subsystem contains a clock which depends on the signals it actually defines
- under-constrained, if the assignment problem failed for the remaining equations
- over-constrained, if any subsystem only constraints signals that have already been determined.

An additional check has to be performed in order to validate the whole system: the clock of any “pre

signal” should be proven equal to the clock of the signal itself. The clock of a given signal can be determined by “summing” the clocks of when constraints that define that signal (union of clocks). The clock of a “pre signal” is the union of the minimal clocks at which that signal is required to be present (i.e., immediate contextual clocks during clock calculus). By “subtracting” (clock difference) both resulting clocks we have to find **empty**. Notice that the problem is decidable since we required full reduction of clock expressions to canonical form. That algorithm not only validates the original system, it also returns its *constrained dataflow representation* which can be used to generate efficient code.

4 Application to Modelica

One can notice that Modelica's sample expressions yield boolean values that are quite exclusively used to activate discrete equations. The reason is that in Modelica when clauses require a test to be performed to activate/deactivate associated equations. But most of the time, that test is useless: it only makes sense to “activate equations from the outside”. Indeed, a when clause which activation constraint only depends on pure events (as ideally generated by sample) would not need to check anything: the activation/deactivation logic would be “lifted” in the control flow. It follows that when clauses can be made more general and efficient by only depending on clocks instead of boolean signals⁷: pure event-based activations do no longer lead to any test in the generated code. That is particularly interesting in presence of external event sources, which in Modelica currently lead to the generation of “event loops” that are extremely resource-consuming (and that eventually require dynamic synchronization with the source). Our proposal avoids the generation of those expensive loops.

In consequence of the above remarks, we propose to equip Modelica with a new type: Clock, the type of clocks (i.e., sequences of logical time events). We also propose to change the semantics of sample expressions, so that they now denote pure event generators⁸. Here is a modification using clocks of the

⁷ This is the reason why the name “when” has been coined historically in synchronous languages such as Signal.

⁸ It would be possible to make sample generate booleans *and* events simultaneously to avoid too many compatibility issues, but, since uses of values yielded by sample as regular booleans seems highly suspicious in our opinion, a more restrictive definition of sample would help to find abusive use of event-generating expres-

original model class TestCounters defined above:

```
connector Out = output Clock;
```

```
model EventSource "Simple event source"
```

```
  parameter Real t0, T;
  Out out;
```

```
equation
```

```
  out = sample(t0, T);
```

```
end EventSource;
```

```
connector In = input Clock;
```

```
model Counter "Simple event counter"
```

```
  parameter Integer n;
  In ins[n];
  Integer count;
```

```
initial equation
```

```
  count = 0;
```

```
equation
```

```
  when ins then
```

```
    count = pre(count) + 1;
```

```
  end when;
```

```
end Counter;
```

```
model TestCounters
```

```
  EventSource src[2](t0 = { 0, 3 },
  T = { 1, 2 });
```

```
  Counter cnt(n = 2);
```

```
equation
```

```
  connect(src.out, cnt.ins);
```

```
end TestCounters;
```

As shown in previous section, an instance of the above model can be represented in our intermediate language as:

```
let c[1] = sample(t0[1], T[1]) in
let c[2] = sample(t0[2], T[2]) in
count = 0 when initial ||
count = pre(count) + 1 when c[1] v
c[2]
```

Given a translation from the original Modelica code to our system description language, we can proceed with *static scheduling* of equations by applying the algorithm proposed above. If the system does not contain any implicit variable, the result is simply composed of several sequences of assignments, activated by *primary clocks* (i.e., source clocks of the system). It is the case in our simple example:

```
when initial:
```

```
count := 0
```

```
when sample(t0[1], T[1]) v sample(t0[1], T[1]):
```

```
count := pre(count) + 1
```

The above representation of the system reads as follow:

- at initialization, assign 0 to the *program variable* count
- on each activation scheduled by **sample**(t0[1], T[1]) v **sample**(t0[1], T[1]), assign the last value held by count plus one to count.

We have derived *sequences of assignments* from the functional specification expressed in our system description language.

5 Example

We consider a well-known example in the Hybrid Systems and Control Theory literature (we will use the version presented in [BK09]). Here we consider the system consisting of a water tank, where water arrives at a variable rate $w_i(t) \geq 0$ through one pipe and leaves through another one at the rate controlled by a valve (cf. Figure 5). The output pipe has a maximal throughput capacity C , and the valve position is given by $0 \leq v(t) \leq 1$. Thus, the actual throughput of the output pipe at the moment t is $C \cdot v(t)$. The valve is controlled by a sensor measuring the level l of water in the tank, which aims at keeping this level in a given interval $[L1, L2]$. For simplicity we assume that there is always enough water in the tank to saturate the output pipe and that the incoming flow does not exceed the output pipe's capacity, i.e. $\max w_i(t) \leq C$.

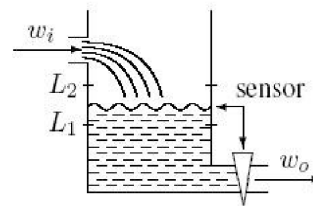


Figure 5: Tank with regulated valve

The transfer function of the complete system has the input space $In = R^+$ (incoming flow rate) and the output space $Out = R^+ \times +$ (output flow rate and current water level in the tank). This system can be modeled as a composition of three sub-systems (see Figure 6).

sions in existing programs.

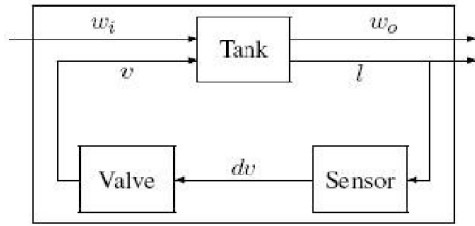


Figure 6: Block-diagram representation of the regulated tank

1. The Tank, taking on input current values of the incoming flow w_i and the position of the valve v and computing the corresponding output flow w_o and water level l from the equations

$$dl = w_i - w_o$$

$$w_o = C \cdot v$$

The corresponding transfer function has the input and output spaces $In_T = R^+ \times [0, 1]$ and $Out_T = R^+ \times R^+$.

2. The Sensor, taking on input the water level l and computing the corresponding valve position adjustment dv from some given equation, e.g.:

$$dv = \text{sign}\left(l - \frac{L1 + L2}{2}\right)$$

The corresponding transfer function has the input and output spaces $In_S = R^+$ and $Out_S = [-1, 1]$.

3. The Valve, taking on input the adjustment dv and providing on output the corresponding value v .

The corresponding transfer function has the input and output spaces $In_V = [-1, 1]$ and $Out_V = [0, 1]$.

This kind of physical models is straightforward to define as a *continuous-time* Modelica model:

```
// Connector definitions
```

```
connector RealInput = input Real;
connector RealOutput = output Real;
```

```
// Submodel definitions
```

```
model Tank
  parameter Real C;
  RealInput wi;
  RealInput v;
  RealOutput wo;
  RealOutput l;
```

```
equation
  wo = C * v;
  der(l) = wi - wo;
end Tank;
```

```
model Sensor
  parameter Real L1, L2;
  RealInput l;
  RealOutput dv;
```

```
equation
  dv = sign(l - (L1 + L2) / 2);
end Sensor;
```

```
model Valve
  RealInput dv;
  RealOutput v;
```

```
equation
  /* In order to preserve the range
  of v, we
  have to constrain the values
  of its derivative */
  der(v) =
    if pre(v) <= 0 then max(dv, 0)
    elseif pre(v) >= 1 then min
      (dv, 0)
    else dv;
end Valve;
```

```
model Source "Sinusiodal source of
flow"
```

```
  constant Real PI = acos(-1);
  parameter Real W0, f;
  RealOutput wo;
equation
  wo = W0 * (0.5 * sin(2 * PI * f *
time) + 0.5);
end Source;
```

```
model TankSensorValve "Agregation
of a tank, a sensor and a valve"
```

```
  RealInput wi;
  RealInput wo;
  Tank tank(C=10, l(start=1.5));
  Sensor sensor(L1=1, L2=2);
  Valve valve(v(start=0));
equation
  connect(wi, tank.wi);
  connect(tank.l, sensor.l);
  connect(sensor.dv, valve.dv);
  connect(valve.v, tank.v);
  connect(tank.wo, wo);
end TankSensorValve;
```

```
// Example of use

model M "A simple use of above models"
  TankSensorValve tankSensorValve(
    tank(C=10, l(start=1.5)),
    sensor(L1=1, L2=2),
    valve(v(start=0));
  Source source(W0=5, f=0.25);
equation
  connect(source.wo, tankSensorValve.wi);
end M;
```

Simulating the above model gives the results in Figure 7.

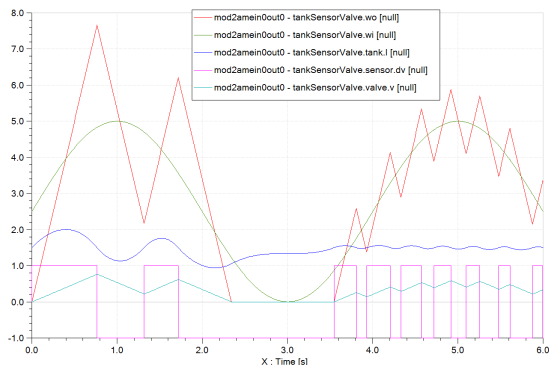


Figure 7: Result of the simulation of a continuous-time model of regulated tank

A discrete-time specification of the regulated tank model would offer several advantages over the continuous-time one: it does not require sophisticated solvers to compute simulation results and it may run far faster than the high-fidelity version (the price to pay when using pure discrete-time models is, most of the time, poor accuracy). Here is a new specification in discrete-time Modelica of the submodels needed to build the final model (changes with respect to original version are in red):

```
// Connector definitions

connector RealInput = input Real;
connector RealOutput = output Real;

//useful constants

constant Real MILLI2SEC = 0.001;
constant Real MILLI_PERIOD = 20;
constant Real STRETCH = 400;
constant Real STEP = MILLI_PERIOD *
MILLI2SEC / STRETCH;
```

```
// Submodel definitions

model Tank
  parameter Real C;
  RealInput wi;
  RealInput v;
  RealOutput wo;
  RealOutput l;
equation
  wo = C * v;
  l = pre(l) + (wi - wo) * STEP;
end Tank;

model Sensor
  parameter Real L1, L2;
  RealInput l;
  RealOutput dv;
equation
  dv = sign(l - (L1 + L2) / 2);
end Sensor;

model Valve
  RealInput dv;
  RealOutput v;
equation
  /* In order to preserve the range
  of v, we
  have to constrain the values
  of its derivative */
  v =
  pre(v) +
  (if pre(v) <= 0 then max(dv, 0)
  elseif pre(v) >= 1 then min
  (dv, 0)
  else dv) * STEP;
end Valve;

model Source "Sinusiodal source of
flow"
  constant Real PI = acos(-1);
  parameter Real W0, f;
  RealOutput wo;
  /* We explicitly define time as a
  discrete signal */
  protected Real time;
  initial equation
  time = 0;
equation
  when sample(0, STEP) then
    time = pre(time) + STEP;
  end when;
  wo = W0 * (0.5 * sin(2 * PI * f *
time) + 0.5);
end Source;
```

```

model Delay "Ideal delay"
  RealInput x;
  RealOutput zx;
equation
  zx = pre(x);
end Delay;

model TankSensorValve "Agregation
of a tank, a sensor and a valve"
  RealInput wi;
  RealInput wo;
  Tank tank(C=10, l(start=1.5));
  Sensor sensor(L1=1, L2=2);
  Valve valve(v(start=0));
  Delay delay(zx(start=0));
equation
  connect(wi, tank.wi);
  connect(tank.l, sensor.l);
  connect(sensor.dv, valve.dv);
  connect(valve.v, delay.x);
  connect(delay.zx, tank.v);
  connect(tank.wo, wo);
end TankSensorValve;

```

Notice the use of difference equations instead of differential ones and also the use of a when clause that provides the “main clock” exported by the Source model class. A new model class is necessary (Delay) to break the algebraic loop (controller feedback) by inserting a delay into it, otherwise the model is not schedulable. The Sensor model class is kept unchanged: that is not surprising since it only involves algebraic equations. Thanks to activation inheritance, model instances connected to Source will be activated by the source's clock. Here is the system of equations resulting from instantiating model class M defined above, expressed in our system representation language:

```

m.tankSensorValve.tank.l = 1.5 when initial ||
m.tankSensorValve.tank.wo = 10 *
m.tankSensorValve.tank.v ||
m.tankSensorValve.tank.l =
pre(m.tankSensorValve.tank.l) +
(m.tankSensorValve.tank.wi -
m.tankSensorValve.tank.wo) * 5e-5 ||
m.tankSensorValve.sensor.dv =
sign(m.tankSensorValve.sensor.l - 1.5) ||
m.tankSensorValve.valve.v = 0 when initial ||
m.tankSensorValve.valve.v =
pre(m.tankSensorValve.valve.v) + (if
pre(m.tankSensorValve.valve.v) <= 0 then
max(m.tankSensorValve.valve.dv, 0) elseif
pre(m.tankSensorValve.valve.v) >= 1 then
min(m.tankSensorValve.valve.dv, 0) else
m.tankSensorValve.valve.dv) * 5e-5 ||
m.tankSensorValve.delay.zx = 0 when initial ||
m.tankSensorValve.delay.zx =
pre(m.tankSensorValve.delay.x) ||

```

```

m.tankSensorValve.wi = m.tankSensorValve.tank.wi ||
m.tankSensorValve.tank.l = m.tankSensorValve.sensor.l
||
m.tankSensorValve.sensor.dv =
m.tankSensorValve.valve.dv ||
m.tankSensorValve.valve.v =
m.tankSensorValve.delay.x ||
m.tankSensorValve.delay.zx =
m.tankSensorValve.tank.v ||
m.tankSensorValve.tank.wo = m.tankSensorValve.wo ||
m.source.time = 0 when initial ||
m.source.time = pre(m.source.time) + 5e-5 when
sample(0, 5e-5) ||
m.source.wo = 5.0 * (0.5 * sin(1.5707963268 *
m.source.time) + 0.5) ||
m.source.wo = m.tankSensorValve.wi

```

Sorting that system gives the following assignments:

```

when initial:
m.tankSensorValve.tank.l := 1.5
m.tankSensorValve.valve.v := 0
m.tankSensorValve.delay.zx := 0
m.source.time := 0

when sample(0, 5e-5):
m.source.time := pre(m.source.time) + 5e-5
m.source.wo :=
5.0 * (0.5 * sin(1.5707963268 * m.source.time) +
0.5)
m.tankSensorValve.wi := m.source.wo
m.tankSensorValve.tank.wi := m.tankSensorValve.wi
m.tankSensorValve.delay.zx :=
pre(m.tankSensorValve.delay.x)
m.tankSensorValve.tank.v :=
m.tankSensorValve.delay.zx
m.tankSensorValve.tank.wo :=
10 * m.tankSensorValve.tank.v
m.tankSensorValve.tank.l :=
pre(m.tankSensorValve.tank.l) +
(m.tankSensorValve.tank.wi -
m.tankSensorValve.tank.wo) * 5e-5
m.tankSensorValve.sensor.l :=
m.tankSensorValve.tank.l
m.tankSensorValve.sensor.dv :=
sign(m.tankSensorValve.sensor.l - 1.5)
m.tankSensorValve.valve.dv :=
m.tankSensorValve.sensor.dv
m.tankSensorValve.valve.v :=
pre(m.tankSensorValve.valve.v) +
(if pre(m.tankSensorValve.valve.v) <= 0 then
max(m.tankSensorValve.valve.dv, 0)
elseif pre(m.tankSensorValve.valve.v) >= 1 then
min(m.tankSensorValve.valve.dv, 0)
else m.tankSensorValve.valve.dv) * 5e-5
m.tankSensorValve.delay.x :=
m.tankSensorValve.valve.v
m.tankSensorValve.wo := m.tankSensorValve.tank.wo

```

The sequences of assignments we have obtained above can be used to feed a real-time embedded code generator. It can be noticed that it can be statically reduced to yield a final code that is both minimal and reliable. This example has shown how Modelica, if equipped with clocks, would lead to reliable and composition-friendly models: we would get the expected behavior by simply connecting generic models, without further adjustments.

6 Conclusions

In this paper, we have shown how the introduction of clocks solves the issues encountered in event-based models written in Modelica. We have also shown that clocks could be harmoniously integrated into the language without compromising simplicity nor expressiveness, on the contrary: models could be made more generic thanks to the modular-friendly aspects of clock calculus (which would help a lot in the design of industrial-strength libraries) and, since it would be possible to express more subtle relationships between event sources, development, debugging and maintainance of models involving discrete-time aspects would be easier.

The Modelica community, in the course of the Modelica 4 design process, is going to consider the problem of synchrony. We hope that modular aspects and expressiveness resulting from the introduction of a full clock calculus will be retained as key features of the new language.

7 Acknowledgements

Many thanks to Ramine Nikoukhah for having pointed out issues related to synchrony and for having been the first to suggest having a look at the concept of clock in the context of Modelica. I would also like to thank Simon Bliudze and Marc Pouzet for interesting discussions about the theoretical aspects of hybrid systems.

References

- [1] R. Nikoukhah, *Activation Inheritance in Modelica*, EOOLT, 2008
- [2] R. Nikoukhah, S. Furic, *Synchronous and Asynchronous Events in Modelica: Proposal for an Improved Hybrid Model*, 6th international Modelica conference, 2008
- [3] A. Benveniste, P. Le Guernic, and C. Jacquemot, *Programming with events and relations: the Signal language and its semantics*, 1991
- [4] S. Bliudze., D. Krob, *Modelling of Complex Systems: Systems as dataflow machines*, 2009

Effective Version Control of Modelica Models

Peter Harman

deltatheta UK Ltd.

The Technocentre, Puma Way, Coventry, CV1 2TT, UK

peter.harman@deltatheta.com

Abstract

This contribution introduces Converge, a specialized Version Control System client application designed purely for Modelica. Conventional VCS clients and diff tools cannot inform the user what the effect of a single edit has on the model as a whole. Converge compares selected revisions of a model, loading the Modelica code directly from the VCS repository. This paper presents examples of Modelica code where an edit that appears significant in a conventional diff tool can be shown as not so, and an edit that appears insignificant in a conventional diff tool actually has significant changes to the resulting model.

Successfully comparing two revisions of a model requires resolving the types of components, including handling inheritance, imports and redeclarations. It requires handling of equations and component values, and flattening of the model structure.

Converge includes a complete Modelica implementation, and presents the VCS repository to the user with a number of views, including Packages, Inheritance, Dependencies, Annotations, and Components views; and Instance and Equations views that compare the instantiated model. Changes, and whether they affect the model results, are highlighted to the user. This will allow users to understand the development of models over time and to solve problems caused by changes in dependent Modelica libraries.

Keywords: software configuration management; version control; model lifecycle management

1 Introduction

As a textual modeling language Modelica [1] allows typical Software Configuration Management (SCM) practices to be used, in particular version control. An important part of version control is the “diff” functionality, allowing the user to see changes between revisions.

The goal of this paper is to introduce a new tool, called Converge, for comparing revisions of Modelica models in order to locate sources of errors, determine which changes have potential effects on model results, and track changes of models over time. Converge connects directly to a version control repository to access models.

The paper is structured as follows. Section 2 is an introduction to version control systems and their use with Modelica including the current limitations. Section 3 describes Converge and the range of views it provides of a model. Section 4 gives some examples where Converge gives a more realistic view of the changes to the model than traditional diff tools. Section 5 describes the implementation of the software, and Section 6 concludes.

2 Version Control Systems

Version Control Systems (VCS) store program code along with a database of revisions, and via client applications, allow the user to access a file or set of files for any revision. Usually this revision can be specified as a date or a revision number. Traditional VCS such as Subversion (SVN) [2] or CVS [3] use a central repository. There is also a new breed of Distributed Version Control Systems (DVCS) such as Git [4] or Mercurial [5] where each working copy also has its own complete copy of the repository. The

style of VCS used however does not affect the benefits and limitations of using VCS with Modelica.

The user normally interacts with the VCS with a client application. These can be standalone; integrated into a file explorer such as the popular TortoiseSVN [6] client; or integrated into an Integrated Development Environment such as Visual Studio [7], Eclipse [8] or Netbeans [9]. These generally have either a built-in diff application, or can launch an external diff application with a pair of revisions.

Web based repository browser applications allow viewing of the code without a client application, for example the Trac [10] system used by the Modelica Association [11]. These usually do not contain diff functionality, but do allow viewing of “changesets”, which list all files changed and a summary of the changes. These give the user a view of the current state, or the state for a particular revision.

2.1 Diff Tools

The output of a diff tool can be used to interpret the possible effects of changes locally. Within one code file, lines of code inserted, changed or removed can be seen, which for a programming language can be interpreted to see the change in behavior. It doesn't however inform the user, for example, whether dependencies have changed or what the effect of a change of import statement or variable type has on the overall program.

In equation based languages such as Modelica or VHDL-AMS [12] it is rarely even possible to determine locally the change in behavior because the code is not algorithmic.

2.2 Other Limitations of VCS and Modelica

Using version control with a Modelica library is better than not using version control. However there are limitations to the information available to the user from the conventional VCS client.

Loading, editing and saving a Modelica model with a Modelica tool may not preserve exactly the same formatting and whitespace as in the original file.

Annotations have no effect on simulation results; however a large proportion of edits within a Modelica diagram editor will be on the annotations. Each of

these edits will be shown in a diff tool and it is currently up to the user to determine which edits are of significance.

Some operations may appear to be significant and yet have no effect on the overall set of variables and equations. Such an example is illustrated later.

3 Viewing Modelica Revisions in Converge

Converge [13] is a standalone tool designed purely for comparing revisions of Modelica models and packages. In essence it is a specialized VCS client designed purely for Modelica. Successfully comparing two models requires resolving the types of components, including handling inheritance, imports and redeclarations. It requires handling of equations and component values, and flattening of the model structure, and therefore has similar needs to a Modelica modeling tool.

The aim of Converge is to successfully solve the problem of version control of Modelica, overcoming the limitations discussed and allowing “model lifecycle management”. Ultimately the aim is to answer typical questions that arise during development or utilization of Modelica libraries:

- Does this change affect the results?
- Why does my model give different behavior to two weeks ago?
- What are the dependencies of my model and which have changed?

This is done by providing the user with a range of views, both of the package structure, and of an individual class.

3.1 Global Views

The user defines the path from which Modelica code is loaded. Rather than just a directory or set of directories as in a modeling tool, the path can contain multiple version control repositories. All views are a comparison between 2 revisions, which can be selected. One of these can be set as a local working directory, so the comparison is between a working version and a committed revision, or it can be between 2 revisions.

There are two overall views, one for the path and one for the Modelica packages.

3.1.1 Path View

The Path view shows all files available to be loaded on the Modelica path. It is not discriminated between the source of the files, whether they are stored in directories, in a version control repository or a Modelica archive file, the resulting tree is the same.

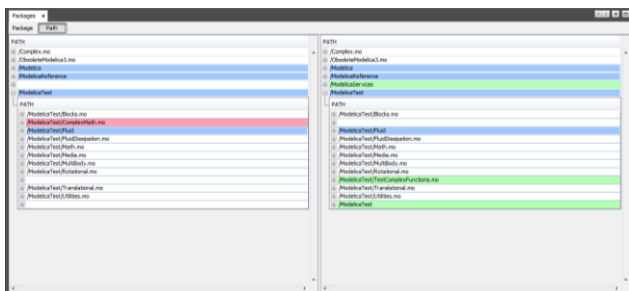


Figure 1: Path View showing revisions of Modelica Standard Library

3.1.2 Packages View

The Packages view shows the hierarchy of Modelica packages as a tree. Similar to the package browser in a Modelica modeling tool, but the two trees side by side show the two revisions being compared. If a class exists in one revision but not the other this is highlighted.

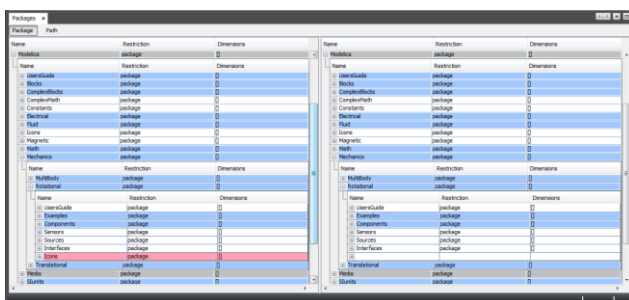


Figure 2: Packages View showing revisions of Modelica Standard Library

3.2 Class Views

An individual Modelica class can be viewed in a number of ways, each designed to visualize a different aspect of the model. Like the global views, these compare the revisions side by side.

3.2.1 Component Structure View

The Structure view shows the components declared in the class. Any components existing in one and not the other will be highlighted, as will differences between individual components. These differences could be the type, dimensions, value or other modifications on the component.

3.2.2 Package Structure View

The Package view shows a tree view of classes declared below the class. Any classes existing in one and not the other will be highlighted. Differences in attributes such as class restriction, dimensions or replaceability will also be highlighted.

3.2.3 Inheritance View

The Inheritance view shows a tree view of classes that the class extends from.

3.2.4 Dependencies View

The Dependencies view shows a tree view of all classes that the class depends on. This could be via inheritance or use as a component. By expanding the tree the user can quickly tell if any dependencies have changed, and changes to key attributes for each class are also highlighted.

3.2.5 Annotations View

In most cases the user will wish to ignore annotations, as these do not affect simulation results. However an Annotations view is included, which shows the user a tree view of all the annotations attached to the class and to components within it.

3.2.6 Instantiated Components View

The Instance view is a powerful way of comparing revisions of a class. The class is shown as a tree of components, both locally declared and inherited. Below each component in the tree is the set of components generated by the flattening of the component.

This allows the user to visualize changes in the resulting flattened model.

3.2.7 Equations View

The Equations view shows a tree with the equations for the class, this can be expanded to see equations from all components in the model, matching the Instance view for components. Equations are determined after modifications are applied but are not simplified and connection equations are not elaborated.

3.2.8 Code View

The Code view is a traditional code diff view, showing the code for each revision with changed lines highlighted.

3.3 Navigation

It is important to be able to quickly navigate through the packages in order to locate a source of a change. Within the global Packages view clicking on any class opens a view of that class. Clicking on a reference to another class within any view, such as Inheritance or Dependencies views, opens a view of it.

Navigating to a different view on the same class is as simple as selecting the tab for the required view.

4 Application Examples

The following examples are to illustrate simple cases where a change that appears significant when comparing code directly causes an insignificant change to the model results, or where a change that appears insignificant comparing code causes a significant change in the model results.

4.1 Editing Diagram

When building Modelica models in a Modelica environment some of the most common operations are in the diagram editor. Some of these have an effect on the results of the model, such as adding or removing components or connections, while some have no effect.

Moving a component in the diagram layer will result in the Placement annotation for that component changing, and the Line annotation for any rerouted connections changing, therefore creating differences on several lines within the model definition. This will be viewed within a VCS client or browser as a significant change, however within Converge it is only shown in the Annotations and Code views.

4.2 Refactoring Inheritance

A common change to models that a library developer may make is moving connectivity and common variables from a model to a partial class, and changing the model to extend from the partial class. This then allows other models to inherit the same connectivity.

```
package Springs
  model SimpleSpring
    Flange_a flange_a;
    Flange_a flange_b;
    Force f;
    Position s_rel;
    parameter Stiffness k=1000
    "Spring rate";
    parameter Distance s_rel0=0 "Un-
    stretched spring length";
    equation
      s_rel = flange_b.s - flange_a.s;
      flange_b.f = f;
      flange_a.f = -f;
      f = c*(s_rel - s_rel0);
    end SimpleSpring;
  end Springs;
```

Code 1: Package before Inheritance Refactoring

```

package Springs
  partial model PartialSpring
    Flange_a flange_a;
    Flange_b flange_b;
    Force f;
    Position s_rel;
  equation
    s_rel = flange_b.s - flange_a.s;
    flange_b.f = f;
    flange_a.f = -f;
  end PartialSpring;

  model SimpleSpring
    extends PartialSpring;
    parameter Stiffness k=1000
    "Spring rate";
    parameter Distance s_rel0=0 "Un-
    stretched spring length";
    equation
      f = c*(s_rel - s_rel0);
    end SimpleSpring;

  model SimpleSpringDamper
    extends PartialSpring;
    Velocity v_rel;
    parameter Stiffness k=1000
    "Spring rate";
    parameter Damping d=10 "Damping
    rate";
    parameter Distance s_rel0=0 "Un-
    stretched spring length";
    equation
      v_rel = der(s_rel);
      f = c*(s_rel - s_rel0) +
d*v_rel;
    end SimpleSpringDamper;
end Springs;

```

Code 2: Package after Inheritance Refactoring

A change such as this will be viewed within a VCS browser, and within the Packages, Class Structure and Inheritance views of Converge, as a significant change. A new class has been added, and the original class has had components removed and an extends-clause added.

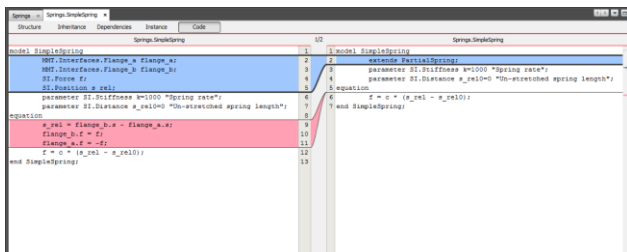


Figure 3: Code View showing Inheritance Example

However, by using the Instance view within Converge, it can be seen that the resulting set of variables has not actually changed. So the user can identify using Converge that this change should not have an impact on the simulation results.

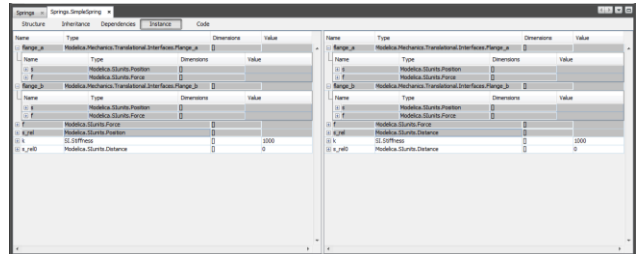


Figure 4: Instance View showing Inheritance Example

4.3 Changing Imports

Changing import statements in a class can change the types of components, cause an error or have no effect at all. Viewed in a traditional diff tool only the line containing the import statement would be highlighted as a change.

The following two models are the same except for a change of import statement. However the components and equations below the “spring” component are different.

```

model MySpringAndMass
  import Spring =
Springs.SimpleSpring;
  Spring spring;
  Mass mass;
  Ground ground;
  equation
    connect(ground.flange,
spring.flange_a);
    connect(spring.flange_b,
mass.flange_a);
  end MySpringAndMass;

```

Code 3: Model before Import Change

```

model MySpringAndMass
  import Spring =
Springs.SimpleSpringDamper;
  Spring spring;
  Mass mass;
  Ground ground;
  equation
    connect(ground.flange,
spring.flange_a);
    connect(spring.flange_b,
mass.flange_a);
end MySpringAndMass;
    
```

Code 4: Model after Import Change

Using the Instance view within Converge, it can quickly be seen what the resulting difference is between the models. Because Converge can resolve the types of components within the model it takes account of the import statements when determining the component tree.

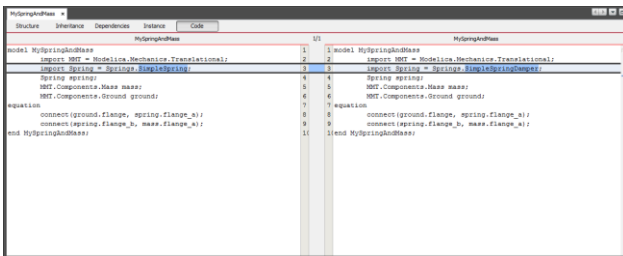


Figure 5: Code View showing Import Example

```

model MySpringAndMass
  replaceable Springs.SimpleSpring
spring constrainedby
Springs.PartialSpring;
  Mass mass;
  Ground ground;
  equation
    connect(ground.flange,
spring.flange_a);
    connect(spring.flange_b,
mass.flange_a);
end MySpringAndMass;
    
```

Code 5: Spring mass model with replaceable spring

```

model MySystem
  MySpringAndMass springMass;
end MySystem;
    
```

Code 6: Model before redeclaration

```

model MySystem
  MySpringAndMass spring-
Mass(redeclare
Springs.SimpleSpringDamper spring);
end MySystem;
    
```

Code 7: Model after redeclaration

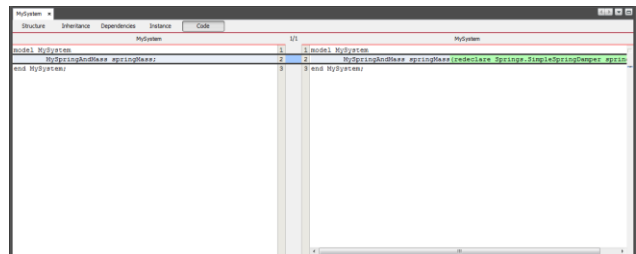


Figure 7: Code View showing Redeclaration Example

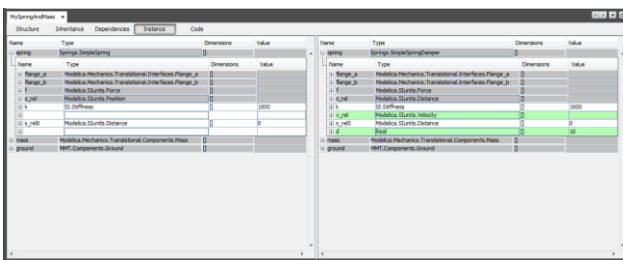


Figure 6: Instance View showing Import Example

4.4 Redclarations

A redeclaration of a component is a small change to the Modelica code that can have a significant change to the flattened model. The spring and mass example from above can be restated as a redeclaration as follows.

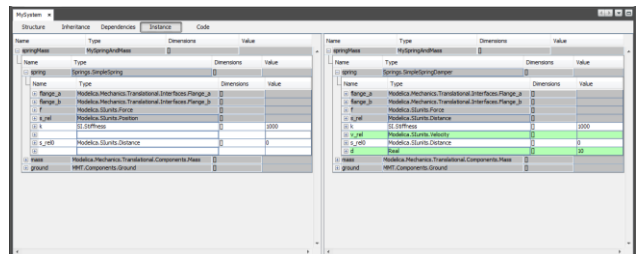


Figure 8: Instance View showing Redeclaration Example

5 Implementation

The ability to load, analyze and navigate the structure of Modelica models, including their instantiated form, is provided by the Modelica SDK [14].

The Modelica Specification [15] defines the mapping between a Modelica package hierarchy and a filesystem. For the general case this is only applicable to a directory structure or a Modelica archive file. The Modelica SDK has an interface that represents this mapping. This defines the methods required to access a hierarchy of Modelica files. Within Converge, implementations of this interface are provided that communicate directly with the VCS system, allowing Modelica models to be loaded for a specified revision without performing a “checkout” operation.

5.1 Status and Limitations

Currently Converge works with Subversion (SVN) repositories. This is being expanded to include a range of version control systems.

Converge is based on a Modelica implementation designed for Modelica 3.x. Libraries containing experimental language features, especially those that change the general syntax or class look-up process, may not give the expected results.

Differences between connections and overconstrained Connections.branch/.root statements are shown in the tool. Connection equations are not generated, including Stream equations and overconstrained branches.

5.2 Issues

It should be stressed here that the analysis of whether a change to a model potentially affects the results has to make the assumption that a change to an annotation has no such effect. Since the introduction of the Embedded Systems section to the specification this is no longer the case, as the mapping to the target system is defined as an annotation.

5.3 Future

Although Converge is not attempting to compile or simulate models, it can still detect sources of errors. Examples of these are changes of names or removal of classes or components that result in failure to find

a class or component, or some cases of incompatible types or dimensions. If such an error occurs in the working version but not in the “head” revision within the VCS then a warning could be issued to the user.

6 Conclusions

In this paper, we have introduced a tool, Converge, for comparison of revisions of Modelica packages within a version control system. This will allow users to understand the development of models over time and to solve problems caused by changes in dependent Modelica libraries.

References

- [1] Modelica, <http://www.modelica.org>
- [2] Collins-Sussman, B., The Subversion Project: Building a Better CVS, Linux Journal, Volume 2002 Issue 94, February 2002
- [3] Morse, T., CVS, Linux Journal, Volume 1996 Issue 21, Jan. 1996
- [4] GIT - Fast Version Control System, <http://git-scm.com>
- [5] O’Sullivan, B., Distributed revision control with Mercurial, Mercurial Project 2007
- [6] TortoiseSVN, <http://tortoisesvn.org>
- [7] Visual Studio, <http://www.microsoft.com/visualstudio/>
- [8] Eclipse, <http://eclipse.org>
- [9] Netbeans, <http://netbeans.org>
- [10] Trac, <http://trac.edgewall.org>
- [11] Modelica Association Trac Instance, <http://trac.modelica.org>
- [12] Christen, E., Bakalar, K., VHDL-AMS, a hardware description language for analog and mixed-signal applications, Circuits and Systems II: Analog and Digital Signal Processing, Volume 26 Issue 10, 1999
- [13] Converge, <http://www.deltatheta.com/products/converge/>
- [14] Harman P., Tiller M. Building Modelica Tools using the Modelica SDK, Modelica 2009
- [15] Modelica Language Specification, Version 3.2, Modelica Association 2010

Automated Simulation of Modelica Models with QSS Methods - The Discontinuous Case -

Xenofon Floros¹ Federico Bergero² François E. Cellier¹ Ernesto Kofman²

¹Department of Computer Science, ETH Zurich, Switzerland
{xenofon.floros, francois.cellier}@inf.ethz.ch

²Laboratorio de Sistemas Dinámicos, FCEIA, Universidad Nacional de Rosario, Argentina
CIFASIS-CONICET
{fbergero, kofman}@fceia.unr.edu.ar

Abstract

This study describes the current implementation of an interface that automatically translates a discontinuous model described using the **Modelica** language into the Discrete Event System Specification (**DEVS**) formalism. More specifically, the interface enables the automatic simulation of a Modelica model with discontinuities in the **PowerDEVS** environment, where the Quantized State Systems (**QSS**) integration methods are implemented. Providing DEVS-based simulation algorithms to Modelica users should extend significantly the tools that are currently available in order to efficiently simulate several classes of large-scale real-world problems, e.g. systems with heavy discontinuities. In this work both the theoretical design and the implementation of the interface are discussed. Furthermore, simulation results are provided that demonstrate the correctness of the proposed implementation as well as the superior performance of QSS methods when simulating discontinuous systems.

Keywords: *OpenModelica, DASSL, PowerDEVS, QSS, discontinuous systems*

1 Introduction

Modelica [8, 9] is an object-oriented, equation-based language that allows the representation of continuous as well as hybrid models using a set of non-causal equations. The Modelica language enables a standardized way to model complex physical systems containing, e.g., mechanical, electrical, electronic, hydraulic, thermal, control, electric power, or process-oriented subcomponents.

Commercial environments, such as Dymola and Scicos, along with open-source implementations, such as OpenModelica [7], enable modeling and simulation of models specified in the Modelica language. All of these tools perform a series of preprocessing steps (model flattening, index reduction, sorting and optimizing the equations) and convert the model to a set of explicit Ordinary Differential Equations (ODEs) of the form $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, t)$. Efficient C++ code is then generated to perform the simulation. Numerical ODE solvers are provided that invoke the right-hand side evaluation of the ODEs at discrete time steps t_k , in order to compute the next value of the state vector \mathbf{x}_{k+1} . Thus, the commonly used simulation environments make use of **time slicing**, i.e., their underlying simulation algorithms are based on time discretization rather than state quantization.

In the end of the nineties, a new class of algorithms for numerical integration based on **state quantization** and the DEVS formalism was introduced by Zeigler [17]. Improving the original approach of Zeigler, Kofman developed a first-order non-stiff Quantized State System (QSS) algorithm in 2001 [14], followed later by second- and third-order accurate non-stiff solvers, called QSS2 [10] and QSS3 [13], respectively. Finally, the family of QSS methods has been further expanded and includes now also stiff system solvers (LIQSS [15]) as well as solvers for marginally stable systems (CQSS [3]). QSS methods have been theoretically analyzed to exhibit nice stability, convergence, and error bound properties, [4, 13, 14], and in general come with several advantages over classical approaches.

Most of the classical methods that use discretization of time, need to have their variables updated in a **synchronous** way. This means that the variables

that show fast changes are driving the selection of the time steps. In a stiff system with widely-spread eigenvalues, i.e., with mixed slow and fast subsystems, the slowly changing state variables will have to be updated much more frequently than necessary, thus increasing substantially the computation time of the simulation. On the other hand, the QSS methods allow for **asynchronous** variable updates, allowing each state variable to be updated at its own pace, and specifically when an event triggers its evaluation. Furthermore, as most systems are sparse, when a state variable x_i changes its value, it suffices to evaluate only those components of \mathbf{f} that depend on x_i , allowing for an additional **significant reduction of the computational costs**. In [5], comparisons have been performed between the standard DASSL solver and QSS3 on synthetically generated sparse linear models that demonstrate the superiority of QSS methods, theoretically expected, when simulating sparse systems.

Another advantage of QSS methods concerns the simulation of **discontinuous** systems with frequent switching behavior, e.g. power electronic circuits. Standard Dymola and OpenModelica software handle discontinuities by means of **zero-crossing functions** that need to be evaluated at each step. When any of them changes its sign, the solver knows that a discontinuity occurred. Then an iterative process is initiated to detect the exact time of that event. In contrast, QSS algorithms offer **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**. Since in a real-time simulation the computational load per unit of real time must be controllable, Newton iterations are usually not acceptable.

Finally, DEVS methods [12] provide a formal unified framework for the simulation of **hybrid systems**, where continuous-time, discrete-time, and discrete-event models can coexist as subcomponents of a single model.

Therefore, **QSS methods** and the principle of **state quantization** appear promising in the context of simulating certain classes of real-world problems. However, in order to simulate a system with QSS methods in the PowerDEVS environment [2], the user needs to have a thorough understanding of DEVS systems. More specifically, the model needs to be manually converted to an explicit ODE form, dependencies between subsystems need to be identified, and the corre-

sponding DEVS structure needs to be provided. Even if a user possesses the required knowledge to do so, this approach is feasible only for very small systems.

PowerDEVS does not support object-oriented modeling, whereas Modelica does. For all these reasons, it is much more convenient for a user to formulate models in the Modelica language than in PowerDEVS.

This work aims to bridge the gap between the powerful object-oriented modeling platform of Modelica on the one hand, and the equally powerful simulation platform of PowerDEVS on the other. In [5], a first version of the interface between OpenModelica and PowerDEVS, for systems without discontinuities, has been presented and analyzed. This study extends the previously discussed interface to include discontinuous models and brings us one step closer to the final goal, enabling a modeler to formulate arbitrary models in the Modelica language, while automatically simulating them in PowerDEVS.

1.1 Relevance of Work

In [5] a first version of an interface between OpenModelica and PowerDEVS for non-stiff and non-discontinuous models has been presented. The current article extends upon [5] to enable the simulation of discontinuous models. To our knowledge there exist no other approaches that automatically translate Modelica models to the DEVS formalism. Research efforts have been reported that implemented Modelica libraries allowing DEVS models to be formulated within a Modelica environment [1, 16], but these approaches require from the users to understand the DEVS framework, as they would have to model their system in the DEVS formalism in order to make use of these libraries.

Furthermore, this is the first work offering a comparison of the run-time efficiency and simulation accuracy of various solvers (DASSL, Radau IIa, Dopri45) in Dymola and OpenModelica against QSS methods. In earlier publications describing QSS methods [10, 11, 12, 14], there can be found examples that demonstrate the superiority of the run-time efficiency of QSS methods, but the comparisons were performed after manual modeling in PowerDEVS directly, i.e. they did not make use of the same original models formulated in Modelica.

In contrast, our approach enables a Modelica user to simulate a Modelica model using QSS solvers without any explicit manual transformation. Addi-

tionally, it allows for the automatic transformations of large-scale models to the DEVS formalism, which is a difficult if not unfeasible task even for experts in DEVS modeling.

The article is organized as follows: Section 2 provides a brief introduction of the QSS methods. Section 3 describes theoretically what is needed in order to simulate a Modelica model with discontinuities employing the QSS algorithms. In Section 4, the actual implementation of the interface between OpenModelica and PowerDEVS is presented. Section 5 describes the simulation results comparing the various solvers in Dymola and the OpenModelica runtime environment with the QSS methods as implemented in PowerDEVS. Finally, Section 6 concludes this study, lists open problems, and offers directions for future work.

2 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, [14], approximates the ODE of 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**. The quantized state $q_i(t)$ only changes when it differs from $x_i(t)$ by more than ΔQ_i . In QSS, the quantized states $\mathbf{q}(t)$ follow 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, $\mathbf{x}(t)$, 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 [10], 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 [13], the quantized states follow piecewise parabolic trajectories, while the states themselves exhibit piecewise cubic trajectories.

3 Simulation of Discontinuous Modelica Models with QSS Methods

In this section we shall describe a potential way to simulate a Modelica model using QSS methods. For simplicity, we shall assume that the model is described by an ODE system, but we note that the interface successfully handles DAE systems as well. Let us write Eq. 2 expanded to its individual component equations, forgetting for a while the discontinuous part:

$$\begin{aligned} \dot{x}_1 &= f_1(q_1, \dots, q_n, t) \\ &\vdots \\ \dot{x}_n &= f_n(q_1, \dots, q_n, t) \end{aligned} \quad (4)$$

If we consider a single component of Eq. 4, we can split it into two equations:

$$q_i = Q(x_i) = Q\left(\int \dot{x}_i dt\right) \quad (5)$$

$$\dot{x}_i = f_i(q_1, \dots, q_n, t) \quad (6)$$

3.1 Accounting for Discontinuities

Discontinuities in dynamical systems are closely related to the notion of events. We can distinguish two types of events, time events and state events.

3.1.1 Time Events

Time events correspond to changes of states as a function of the built-in continuously evolving variable **time**. Such events can be scheduled in advance, since it is possible to predict the point in time when they occur. Time events in Modelica are specified basically in two ways [6]:

- With a conditional discrete-time expression that contains the variable *time* (e.g. in a when-statement) of the form:
time \geq *discrete-time expression*, e.g. $t \geq t_e$

- With a periodic *sample* statement of the form: $sample(first, interval)$ that triggers events at predefined time instants.

The first case can be taken care of by formulating a zero-crossing function of the form:

$$g(t) = t - t_e$$

When $g(t)$ crosses through zero, an event should be produced. We shall see later, which DEVS blocks need to be defined to generate the events. The *sample()* statement can be handled easily by adding a dedicated DEVS atomic model that provokes events at the predefined time points.

3.1.2 State Events

State events are related to discrete changes in the state variables during the simulation as a function of other state variables reaching some threshold value. Therefore, they cannot be scheduled in advance. A state event can be specified by means of *when* or *if-then-else* statements involving one or more state variables. When a model is compiled by either OpenModelica or Dymola, state events are translated into **zero-crossing functions** of the form $g_i(\mathbf{x}, t)$. During the execution of the simulation the zero-crossing functions are being constantly monitored and when function $g_i(\cdot)$ crosses through zero, a discontinuity is detected and handled accordingly. Therefore, we can directly exploit the zero-crossing functions generated by OpenModelica to identify state events in an identical fashion as with time events. All we need is a **Static Function** block evaluating the zero-crossing function and a **Zero-Cross Detection** block that detects when a zero-crossing takes place.

3.2 DEVS structure

The DEVS formalism [17] allows to describe both the continuous and discontinuous parts of the model via a coupling of simpler DEVS atomic models. More specifically, we need to define:

- A **Quantized Integrator** block (Eq. 5) that takes as input the derivative \dot{x}_i and outputs q_i .
- A **Static Function** block that receives the sequence of events, q_1, \dots, q_n , and calculates the sequence of state derivative values, \dot{x}_i (Eq. 6). The same block can be used for the evaluation of the zero-crossing functions $g_i(\cdot)$

- A **Cross-Detection** block that receives as input the evaluated zero-crossing function and generates an output event when its input crosses through zero.

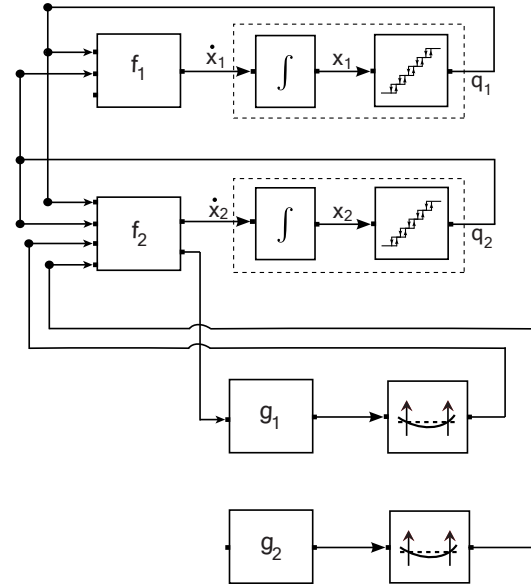


Figure 1: Coupled DEVS model for QSS simulation of a discontinuous model with 2 states and 2 zero-crossing functions $g_1(\cdot)$ and $g_2(\cdot)$.

Therefore, we can simulate a Modelica discontinuous model using a coupled DEVS model consisting of the blocks described above. A block diagram representing the final DEVS model for an example system with 2 state variables and 2 zero-crossing functions is shown in Fig. 1.

4 OpenModelica to PowerDEVS (OMPD) Interface

This section describes the work done to enable the simulation of Modelica models in PowerDEVS using QSS algorithms. The current version of the interface does not yet support **when** clauses and **sample** statements.

4.1 What is Needed by PowerDEVS

Let us first concentrate on what PowerDEVS requires in order to perform the simulation of a Modelica model. As depicted in Fig. 1, an essential component of a PowerDEVS simulation is the graphical structure. In PowerDEVS, the structure is provided in the

form of a dedicated **.pds structure file** that contains information about the blocks (nodes) of the graph as well as the connections (edges) between those blocks. More specifically, we need to add in the structure:

- A **Quantized Integrator** block for each state variable with \dot{x}_i as input and q_i as output.
- A **Static Function** block for each state variable that receives as input the sequence of events, q_1, \dots, q_n , and calculates $\dot{x}_i = f_i(\mathbf{q})$.
- A **Static Function** block for each one of the zero-crossing functions $g_i(\cdot)$ generated by OpenModelica that receives as inputs the dependencies of $g_i(\cdot)$ and evaluates the function in the output port.
- A **Cross-Detection Block** block after each one of the zero-crossing static functions. The cross-detection block outputs an event if a zero-crossing has been identified.
- A **connection (edge)** is added between two blocks if and only if there is a dependence between them.

Having correctly identified the DEVS structure, we need to specify what needs to be calculated inside each of the static function blocks. The different blocks need to have access to different pieces of information.

In the current implementation, a **.cpp code file** is generated that contains the code and parameters for all blocks in the structure. The generated code file contains the following information:

- For each **Quantized Integrator** block, the initial condition, error tolerance, and integration method (QSS, QSS2, QSS3).
- For each **Static Function**, the equations/expressions needed in order to calculate the derivative of each state variable in the system. Furthermore, the desired error tolerance is provided together with a listing of all input and output variables of the specific block. If the static function represents a zero-crossing then it contains the respective function $g_i(\cdot)$.

4.2 What is Provided by OpenModelica

In Section 4.1, we described what PowerDEVS expects in order to perform the simulation. Our work

focuses on an automatic way to simulate Modelica models using the QSS methods in PowerDEVS. Therefore, the PowerDEVS simulation files should be automatically generated exploiting the information contained in the Modelica model supplied as input. Luckily, existing software used to compile Modelica models, such as Dymola or OpenModelica, produces simulation code that contains all information needed by PowerDEVS. Thus, we were able to make use of an existing Modelica environment by modifying the existing code generation modules at the back end of the compiler to produce the files needed by PowerDEVS.

This work is based on modifying the OpenModelica Compiler (OMC), since it is open-source and has a constantly growing contributing community. OMC takes as input a Modelica source file and translates it first to a flat model. The flattening consists of parsing, type-checking, performing all object-oriented operations such as inheritance, modifications, etc. The flat model includes a set of equation declarations and functions with all object-oriented structure removed. Then index reduction is performed on the set of model equations in order to remove algebraic dependence structures between state variables. The resulting equations are then analyzed, sorted in Block Lower Triangular (BLT) form, and optimized. Finally, the code generator at the back end of OMC produces C++ code that is then compiled. The resulting executable is used for the simulation of the model.

The information needed to be extracted from the OMC compiler is contained mainly in the DLOW structure, where the following pieces of information are defined:

- Equations: $E = \{e_1, e_2, \dots, e_N\}$.
- Variables: $V = \{v_1, v_2, \dots, v_N\} = V_S \cup V_R$ where V_S is the set of state variables with $|V_S| = N_S \leq N$ and V_R is the set of all other variables in the model.
- BLT blocks: subsets of equations $\{e_i\}$ needed to be solved together because they are part of an algebraic loop.
- Zero-Crossings: $G = \{g_1, g_2, \dots, g_K\}$.
- Incidence matrix: An $N \times N$ adjacency matrix denoting, which variables are contained in each equation.

The OMPD interface utilizes the above information and implements the following steps:

1. **Equation splitting** : The interface identifies the equations needed in order to compute the derivative $\dot{x}_i = f_i(\mathbf{q})$ for each state variable. Then the split equations can be assigned to static function blocks according to the state derivative evaluation they are involved in.
2. **Mapping split equations to BLT blocks** : The equations are mapped back to BLT blocks of equations in order to be able to generate simulation code for solving linear/non-linear algebraic loops.
3. **Identifying zero-crossing functions** : The zero-crossing functions generated by OMC are extracted and assigned to separate static function blocks.
4. **Constructing generalized incidence matrix** : The $N \times N$ adjacency matrix has to be expanded to include also the zero-crossing functions and the variables involved in them. Thus, it has to be expanded by adding K rows corresponding to the K zero-crossing functions $g_i(\cdot)$. The result is a generalized $K \times N$ adjacency matrix.
5. **Generating DEVS structure** : In order to correctly generate the DEVS structure of the model, the dependencies between the individual DEVS blocks need to be resolved. This is accomplished by employing the generalized incidence matrix to find the corresponding inputs and outputs for each block.
6. **Generating the .pds structure file**: Having correctly produced the DEVS structure for PowerDEVS, outputting the respective .pds structure file is straightforward.
7. **Generating static blocks code** : In this step, the functionality of each static block is defined via the simulation code provided in the **.cpp code file**. Each static block needs to know its inputs and outputs, identified by the DEVS structure, as well as the BLT blocks needed to compute the corresponding state derivatives. The static blocks that are responsible for the discontinuities contain the zero-crossing functions $g_i(\cdot)$ generated by OMC. Then, the existing code generation module of OMC is employed to provide the actual simulation code for each static block, since it has already been optimized to solve linear and non-linear algebraic loops.

8. **Generating the .cpp code file**: The code for the static blocks is output in the .cpp code file along with other needed information.

5 Simulation Results

5.1 Benchmark Framework

In this section, the simulation results obtained using the OMPD interface are presented and discussed. The goal is to compare the run-time efficiency and accuracy of the QSS methods against other simulation software environments. More specifically, we want to compare QSS3 and QSS2 methods in PowerDEVS v2.0 against the DASSL, Radau IIa, and Dopri45 solvers implemented in Dymola v7.4 and the DASSL solver of OpenModelica v1.5.1.

DASSL was chosen as it represents the state-of-the-art multi-purpose stiff DAE solver used by most commercial simulation environments today. **Radau IIa** was included in the comparisons, because a single-step (Runge-Kutta) algorithm is supposed to be more efficient than a multi-step (BDF) algorithm when dealing with heavily discontinuous models, because step-size control is more expensive for the latter methods [4]. Finally, **Dopri45** was chosen, because it is an explicit Runge-Kutta method in contrast to both DASSL and Radau IIa, which are implicit algorithms that may be disadvantaged when simulating non-stiff systems.

As benchmark problems we focused on two real-world systems exhibiting heavily discontinuous behavior, namely a half-way rectifier circuit, modeled graphically with standard Modelica components as depicted in Fig. 2, and the switching power converter circuit provided in Fig. 3.

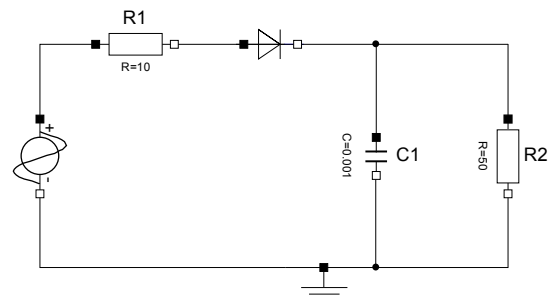


Figure 2: Graphical representation of the half-way rectifier

In order to measure the execution time for each simulation algorithm, the reported simulation time

from each environment was used. Dymola reports CPU-time for integration, OpenModelica reports timeSimulation, and PowerDEVS the elapsed simulation time. To record pure simulation time, the generation of output files was suppressed in all cases. Testing has been carried out on a Dell 32bit desktop with a quad core processor @ 2.66 GHz and 4 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.

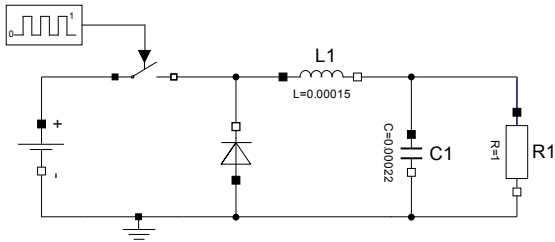


Figure 3: Graphical representation of the switching power converter

Calculating the accuracy of the simulations can only be performed approximately, since the state trajectories in the two 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, Dymola was employed using the default DASSL solver with a very tight tolerance of 10^{-12} and requesting 10^5 output points. Furthermore, in order to verify the accuracy of the reference solution, a second reference solution was computed using QSS3 in PowerDEVS with the tolerance set to 10^{-12} . However, we only report the simulation error against the Dymola solution since the difference between both reference solutions is on the order of 10^{-6} .

To calculate the simulation error, each one of the simulated trajectories was compared against the two reference solutions. To achieve this goal, we forced all solvers to output 10^5 equally spaced points for obtaining simulation trajectories ($\mathbf{t}^{\text{ref}}, \mathbf{y}^{\text{sim}}$) without changing the integration step. Then, the mean absolute error is calculated as:

$$\text{error} = \frac{1}{|t^{\text{ref}}|} \sum_{i=1}^{|t^{\text{ref}}|} |y_i^{\text{sim}} - y_i^{\text{ref}}| \quad (7)$$

In the case of more than one state variables, we report the mean error over all state trajectories.

5.2 Half-Way Rectifier

The half-way rectifier circuit exhibits only one state variable, namely the voltage across the capacitor C1, and the model is simulated during 1 sec. In Fig. 4, the state trajectory calculated with QSS3 and a tolerance of 10^{-4} is depicted. Comparing the simulation results listed in Table 1, the following conclusions can be reached:

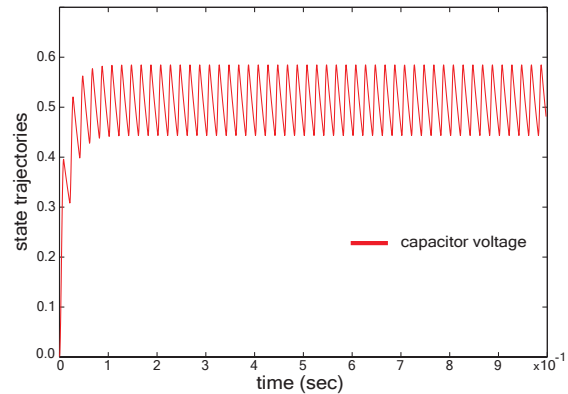


Figure 4: Simulated state trajectories with QSS3 for the half-way rectifier circuit.

There is a **substantial difference in execution efficiency between Dymola and OpenModelica** using the DASSL solver, with Dymola being around 10 times faster than OpenModelica in spite of the fact that both environments make use of the same solver software and even the same root solver (event detection) algorithms. We postulate that this difference is primarily caused by the fact that OMC does not involve **tearing**. Thereby the solution of algebraic loops becomes much less efficient, and also the integration itself suffers, because the number of iteration variables in DASSL equals the number of state variables plus the number of tearing variables. Without tearing, DASSL needs to include all variables appearing inside algebraic loops among the set of its iteration variables.

On the other hand, even though the QSS3 simulations are based on code generated by OMC, we observe that QSS3 is slightly more efficient than DASSL in Dymola. To perform the simulation for an achieved error of the order of 10^{-4} , QSS3 required 0.014 sec while DASSL 0.022 sec. Therefore, **the use of the OMPD interface and the simulation in PowerDEVS employing QSS3 speeds up the simulation by a factor of 20** compared to OpenModelica. It needs to be remarked that it is not fair to compare QSS3 with the DASSL simulation of Dy-

Table 1: This table depicts the simulation results of various algorithms for the half-way rectifier circuit for a requested simulation time of 1 sec. The comparison performed includes required CPU time (in sec) as well as the simulation accuracy relative to the reference trajectory obtained in Dymola.

			CPU time (sec)	Simulation Error
Dymola	DASSL	10^{-3}	0.019	1.45E-03
	DASSL	10^{-4}	0.022	2.35E-04
	Radau IIa	10^{-7}	0.031	2.20E-06
	Dopri45	10^{-4}	0.024	4.65E-05
PowerDEVS	QSS3	10^{-3}	0.014	2.59E-04
	QSS3	10^{-4}	0.026	2.23E-05
	QSS3	10^{-5}	0.041	2.30E-06
	QSS2	10^{-2}	0.242	3.02E-03
	QSS2	10^{-3}	0.891	3.04E-04
	QSS2	10^{-4}	3.063	3.00E-05
OpenModelica	DASSL	10^{-3}	0.265	3.80E-03
	DASSL	10^{-4}	0.281	5.40E-04

mola because of the fact analyzed earlier, namely the lack of tearing in OMC. The solution of the algebraic loops in QSS3 is based on code generated by OMC, and therefore, the inefficiencies in the compilation of the OMC are being propagated to the QSS3 simulation as well. For this reason, **we need to compare the results in PowerDEVS with the ones obtained by OpenModelica and not by Dymola**. However, it is encouraging to see that the improvement achieved over the standard OMC simulation using QSS-based solvers is such that we are able to obtain simulation results that are even more efficient than those obtained using the commercial Dymola environment. If the QSS methods were implemented in Dymola, the simulation results obtained by the QSS methods would once again be considerably faster than the simulation results that Dymola achieves currently.

Performing an internal comparison between the QSS methods, it is obvious that QSS3 is much more efficient than QSS2. This is expected, since the QSS2 solver needs to take smaller steps compared to QSS3 in order to reach the desired accuracy. **Thus, we can conclude that the third-order QSS3 algorithm should be preferred for practical applications.**

For the sake of completeness we included in the comparison two more solvers included in the Dymola environment, Radau IIa and Dopri45. **Radau IIa** is an implicit variable-step Runge-Kutta method of order 5, while **Dopri45** is an explicit step-size controlled Runge-Kutta algorithm of order 5. For this specific example, Radau IIa failed to provide correct results unless the tolerance was lowered to 10^{-7} .

Radau IIa with a less tight tolerance tries to utilize larger integration steps and, apparently, misses many of the events, i.e. the event localization employed by Dymola is not robust (conservative) enough. It needs to be noted further that the problem got considerably worse between Dymola 6 and Dymola 7, i.e., whereas Radau IIa missed a few events in Dymola 6, it misses many more events in Dymola 7. This is a quite serious issue that the Dynasim company should look into. The same problem was observed for Dopri45 as well, when the tolerance was set to 10^{-3} . Due to these problems, both Runge-Kutta algorithms require CPU times comparable to that needed by the standard DASSL solver, i.e., the inherent advantages of the single-step algorithms over a multi-step technique in dealing with heavily discontinuous models could not be exploited due to the inability of their current implementation to detect events reliably.

5.3 Switching Power Converter

The switching power converter exhibits two state variables, namely the current through the inductor L1 and the voltage across the capacitor C1. From Fig. 3, we see that there is a square wave source block that, when implemented directly, would call for use of a sample block. As the sample block has not yet been implemented in the interface, we worked around this problem by replacing the square wave source by a second-order marginally stable time-invariant system described by:

```
model SquareWaveGenerator
  Real x1(start=0.0);
```


Table 2: This table depicts the simulation results of various algorithms for the switching power converter circuit for a requested simulation time of 0.01 sec. The comparison performed includes required CPU time (in sec) as well as the simulation accuracy relative to the reference trajectory obtained in Dymola.

			CPU time (sec)	Simulation Error
Dymola	DASSL	10^{-3}	0.051	1.82E-04
	DASSL	10^{-4}	0.063	7.18E-05
	Radau IIa	10^{-3}	0.064	1.11E-07
	Radau IIa	10^{-4}	0.062	1.11E-07
	Dopri45	10^{-3}	0.049	6.38E-06
	Dopri45	10^{-4}	0.047	9.76E-06
PowerDEVS	QSS3	10^{-3}	0.049	1.41E-03
	QSS3	10^{-4}	0.062	1.68E-05
	QSS3	10^{-5}	0.250	8.96E-06
OpenModelica	DASSL	10^{-3}	50.496	-
	DASSL	10^{-4}	1.035	2.62E-02

```

Real x2(start=1.0);
Boolean pulse(start=true);
parameter Real freq=1e4;
equation
  der(x1)=freq*4*x2;
  der(x2)=if (x1<0) then freq*4 else -freq*4;
  pulse=(x1>0);
  idealClosingSwitch.control = pulse;
end SquareWaveGenerator;
    
```

This is worth noting since it adds two more states to the model (x_1, x_2) and increases the computation time since the solver also has to simulate the marginally stable system. The chosen solution is by no means unique. The desired switching behavior could have been coded in many different ways.

The model was simulated for 0.01 sec, and in Fig. 5, the state trajectories calculated using the QSS3 solver with a tolerance of 10^{-4} are plotted. The simulation results for all algorithms under comparison are presented in Table 2.

The conclusions reached in the analysis of the results of the half-way rectifier circuit also hold for the switching power converter circuit as depicted in Table 2. **The QSS3 method performs well compared to the DASSL solver in Dymola, while it outperforms DASSL in OpenModelica and the second-order QSS2.** Radau IIa and Dopri45 simulate correctly even for large tolerance values in this example, but their run-time performance is not significantly better than that of DASSL or QSS3.

For the switching power converter circuit, the simulation errors estimated for DASSL in OpenModelica are quite large. This is suspicious, as it should not be

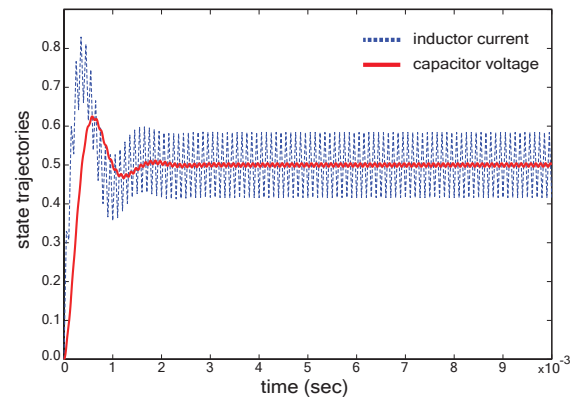


Figure 5: Simulated state trajectories with QSS3 for the switching power converter circuit.

the case. We noticed further that in OpenModelica for a relaxed tolerance of 10^{-3} , the simulation requires a substantial CPU time of 50 sec. The output files generated are also huge, around 500 MB, making it impossible to check if the simulated trajectories are correct or not. There seems to be something wrong with the compilation performed by the OMC in this example, but we cannot make any definite statements regarding this behavior yet.

6 Discussion

6.1 Conclusions

In this article, an extension of the interface between the OpenModelica environment and PowerDEVS presented in [5] is discussed and analyzed. The im-

plemented OMPD interface successfully handles discontinuities and allows to simulate real-world Modelica models with discontinuities using the PowerDEVS simulation software.

Comparisons on two example models were performed, demonstrating the increased efficiency of QSS3 over the standard DASSL solver. The proposed OMPD interface utilizes code generated by the OpenModelica compiler, therefore comparisons must be performed between QSS3 and the DASSL solver of OpenModelica, where we achieve a more than 20-fold decrease in the required CPU time.

Furthermore, comparisons show that the efficiency of QSS3 simulations using code generated by the OMC is comparable to simulations run in Dymola using the built-in DASSL solver, in spite of the fact that Dymola offers much more sophisticated model preprocessing, such as a well-tuned tearing algorithm for the efficient simulation of models involving algebraic loops. Hence we are very optimistic that there would result a significant gain in simulation efficiency if the OMPD interface were to be implemented as part of the back end of the Dymola compiler even in a single-processor implementation, i.e., without exploiting the fact that QSS-based solvers are naturally asynchronous and can therefore be much more easily and elegantly distributed over a multi-core architecture for efficient real-time simulation.

6.2 Future Work

We have shown that the implemented OMPD interface successfully allows a user to simulate Modelica models with discontinuities using PowerDEVS and QSS solvers. However, there still remain open problems that need to be addressed in the future.

As a next step full support for hybrid models needs to be incorporated. This requires the implementation of sample statements and when-clauses. The OMPD interface does not yet support a stiff-system solver. There exist already stiff QSS solvers of orders 1 to 3 [15], which, however, are not yet supported by the interface, because they have not yet been included in the general release of the PowerDEVS software. For this reason, we had to be careful to choose example systems that do not lead to stiff models.

Next, many more models will need to be tested. In particular, we shall need to run all example codes of the Modelica Standard Library that the OMC is able to handle through the interface to verify that the OMPD is capable of handling all models that are

thrown its way.

Finally, we shall make use of the new platform for investigating parallel simulation on a multi-processor architecture. There are many unresolved issues here to be tackled, such as the load balancing problem, i.e., how to optimally distribute the simulation code over a multi-processor architecture. The fact that QSS-based solvers can be easily parallelized does not tell us yet how to optimally make use of that possibility.

7 Acknowledgments

We would like to acknowledge the help and support of the PELAB group at Linköping University and in particular Martin Sjölund, Per Östlund, Adrian Pop, and Prof. Peter Fritzson. Also, we would like to thank Willi Braun of Bielefeld University and Jens Frenkel of TU Dresden for their helpful comments and discussions about the OMC back end.

References

- [1] Tamara Beltrame and François E. Cellier. Quantised State System Simulation in Dymola/Modelica using the DEVS Formalism. In *Modelica*, 2006.
- [2] Federico Bergero and Ernesto Kofman. Powerdevs: a tool for hybrid system modeling and real-time simulation. *SIMULATION*, 2010.
- [3] François Cellier, Ernesto Kofman, Gustavo Migoni, and Mario Bortolotto. Quantized State System Simulation. In *Proceedings of Summer-Sim 08 (2008 Summer Simulation Multiconference)*, Edinburgh, Scotland, 2008.
- [4] François E. Cellier and Ernesto Kofman. *Continuous System Simulation*. Springer-Verlag, New York, 2006.
- [5] Xenofon Floros, François E. Cellier, and Ernesto 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.
- [6] Peter Fritzson. *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*. Wiley-Interscience, New York, 2004.

- [7] Peter Fritzson, Peter Aronsson, Hakan Lundvall, Kaj Nystrom, Adrian Pop, Levon Saldamli, and David Broman. The OpenModelica Modeling, Simulation, and Development Environment. *Proceedings of the 46th Conference on Simulation and Modeling (SIMS'05)*, pages 83–90, 2005.
- [8] Peter Fritzson and Peter Bunus. Modelica - A General Object-Oriented Language for Continuous and Discrete-Event System Modeling and Simulation. In *Annual Simulation Symposium*, pages 365–380, 2002.
- [9] Peter Fritzson and Vadim Engelson. Modelica - a unified object-oriented language for system modeling and simulation. In Eric Jul, editor, *ECOOOP '98 - Object-Oriented Programming*, volume 1445 of *Lecture Notes in Computer Science*, pages 67–90. Springer Berlin / Heidelberg, 1998. 10.1007/BFb0054087.
- [10] Ernesto Kofman. A Second-Order Approximation for DEVS Simulation of Continuous Systems. *Simulation*, 78(2):76–89, 2002.
- [11] Ernesto Kofman. Quantization-Based Simulation of Differential Algebraic Equation Systems. *Simulation*, 79(7):363–376, 2003.
- [12] Ernesto Kofman. Discrete Event Simulation of Hybrid Systems. *SIAM Journal on Scientific Computing*, 25:1771–1797, 2004.
- [13] Ernesto Kofman. A Third Order Discrete Event Simulation Method for Continuous System Simulation. *Latin America Applied Research*, 36(2):101–108, 2006.
- [14] Ernesto Kofman and Sergio Junco. Quantized-state systems: a DEVS Approach for continuous system simulation. *Trans. Soc. Comput. Simul. Int.*, 18(3):123–132, 2001.
- [15] Gustavo Migoni and Ernesto Kofman. Linearly Implicit Discrete Event Methods for Stiff ODEs. *Latin American Applied Research*, 2009. In press.
- [16] Victor Sanz, Alfonso Urquía, François E. Cellier, and Sebastián Dormido. System Modeling Using the Parallel DEVS Formalism and the Modelica Language. *Simulation Modeling Practice and Theory*, 18(7):998–1018, 2010.
- [17] Bernard 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.

Auto-Extraction of Modelica Code from Finite Element Analysis or Measurement Data

The-Quan Pham¹, Alfred Kamusella², Holger Neubert²

¹ OptiY e.K., Aschaffenburg Germany, e-mail: pham@optiy.eu

² Technische Universität Dresden, Institute of Electromechanical and Electronic Design

Abstract

This paper presents a new approach to extract Modelica codes from finite element analysis or measurement data automatically. The finite element model or the real system on the test bench is adaptively sampled while applying the Gaussian process with a few number of model calculations or measurement points. Based on these support points, a meta- or surrogate model of the system is built. Thus, Modelica codes can be generated automatically. These algorithms are implemented in the multidisciplinary design software OptiY[®]. Its application is demonstrated on the example of an electromagnetic actuator.

Keywords: Gaussian Process; Kriging; Surrogate Modeling; Meta Modeling

1 Introduction

The manual modeling of technical systems with network elements is a challenging and time-consuming process. Considerable experience and knowledge on the working principles is necessary. Commercial software systems such as Dymola, Matlab/Simulink, SimulationX, etc. provide ready-to-use model libraries for this approach. However, they do not support the elaboration of an adequate network structure. In order to achieve a sufficient accuracy of the models, an expensive parameter identification has to be performed frequently. This can be achieved by comparing the simulation results with those from experimental investigations and then adjusting the network parameters.

An increasing demand for automatic model generation emerges from this (Fig. 1). Two ways are possible. The first way uses measurement data from a real product or process as a basis of the model. It is not necessary that the working principles or mathemati-

cal relations which describe the system are known. The system is assumed to be a black box.

The second way models the system rigorously, starting from real system geometry, discretizing the system in time and space and using partial differential equations. Mostly, the finite element method is used. Both approaches allow automatic model code generation for the usage in system simulation.

When the equation system is known and the finite element method is applied than a model order reduction (MOR) is possible [7]. However, MOR is not applicable on experimental data or black box models. Alternatively, in such cases, the adaptive Gaussian process [1-5] can be used to generate model codes automatically as we demonstrate in this paper. Meta or surrogate models are derived from these black box systems. Therefore, this approach is more general than MOR.

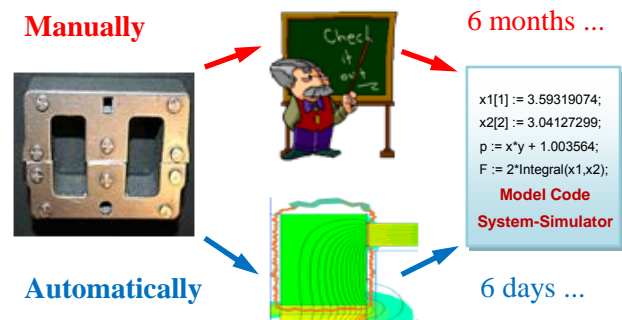


Fig 1: Different modeling approaches

2 Adaptive Gaussian Process

2.1 Gaussian Process

The Gaussian process, also known as Kriging, extends a multivariate normal distribution to infinite dimensionality [1, 2, 3]. The Gaussian process model $Y(\mathbf{x})$ is composed of the global function $f(\mathbf{x})$ and the stochastic process $Z(\mathbf{x})$ representing the deviation from the global function:

$$Y(\mathbf{x}) = \sum_{i=1}^p \beta_i \cdot f_i(\mathbf{x}) + Z(\mathbf{x}).$$

$f(\mathbf{x})$ is the polynomial regression function of any order, β are the unknown regression coefficients, and $Z(\mathbf{x})$ is a stationary stochastic process having mean zero, variance σ^2 and correlation function $R(\bullet)$. \mathbf{x} is a m -dimensional vector of input parameters.

It is assumed, that the training data or support points consist of the simulation or measurement data at the input set $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n$ and that $y(\mathbf{x}_0)$ is the predicted vector. The Gaussian process model implies the points $Y_0 = Y(\mathbf{x}_0)$ and $\mathbf{Y}_n = [Y(\mathbf{x}_1), \dots, Y(\mathbf{x}_n)]^T$ having the multivariate normal distribution:

$$\begin{pmatrix} Y_0 \\ \mathbf{Y}^n \end{pmatrix} \approx N_{n+1} \left[\begin{pmatrix} \mathbf{f}_0^T \\ \mathbf{F} \end{pmatrix} \beta, \sigma_z^2 \begin{pmatrix} 1 & \mathbf{r}_0^T \\ \mathbf{r}_0 & \mathbf{R} \end{pmatrix} \right].$$

$\mathbf{f}_0 = f(\mathbf{x}_0)$ is the $(p, 1)$ -vector of regression function for $Y(\mathbf{x}_0)$. $\mathbf{F} = [f_i(\mathbf{x}_i)]$ is the (n, p) -matrix of regression functions for the training data or support points. $\mathbf{r}_0 = [R(\mathbf{x}_0, \mathbf{x}_1), \dots, R(\mathbf{x}_0, \mathbf{x}_n)]^T$ is the $(n, 1)$ -vector of correlation functions of \mathbf{Y}_n with $Y(\mathbf{x}_0)$, and $\mathbf{R} = R(\mathbf{x}_i, \mathbf{x}_j)$ is the (n, n) -matrix of correlation functions among \mathbf{Y}_n .

Therefore, the best linear unbiased predictor for $Y(\mathbf{x}_0)$ is the mean value of the multivariate normal distribution. It represents the response surface, also known as meta- or surrogate model, of the real system:

$$\hat{Y}(\mathbf{x}_0) = \mathbf{f}_0^T \beta + \mathbf{r}_0^T \mathbf{R}^{-1} (\mathbf{Y}^n - \mathbf{F} \beta).$$

The uncertainty of the predicted value is characterized by the variance of the multivariate normal distribution:

$$\sigma^2 = \sigma_z^2 \left(1 - \mathbf{r}_0^T \mathbf{R}^{-1} \mathbf{r}_0 + (\mathbf{f}_0 - \mathbf{F}^T \mathbf{R}^{-1} \mathbf{r}_0)^T (\mathbf{F}^T \mathbf{R}^{-1} \mathbf{F})^{-1} (\mathbf{f}_0 - \mathbf{F}^T \mathbf{R}^{-1} \mathbf{r}_0) \right).$$

2.2 Correlation Function

The correlation function is the crucial ingredient in the Gaussian process predictor because it contains assumptions about the function to be predicted. It interpolates between support points in which its value smoothly changes between 0 and 1.

Several stationary correlation functions have been investigated to approximate a lot of real functions or systems [1]:

- Gamma exponential

$$R(\mathbf{x}_1, \mathbf{x}_2) = \exp \left\{ - \sum_{i=1}^m w_i^\gamma \cdot |\mathbf{x}_1 - \mathbf{x}_2|^\gamma \right\}$$

- Matérn class

$$R(\mathbf{x}_1, \mathbf{x}_2) = \frac{1}{\Gamma(\nu) 2^{\nu-1}} \left(\frac{2\sqrt{\nu} |\mathbf{x}_1 - \mathbf{x}_2|}{\theta} \right)^\nu K_\nu \left(\frac{2\sqrt{\nu} |\mathbf{x}_1 - \mathbf{x}_2|}{\theta} \right)$$

- Rational quadratic

$$R(\mathbf{x}_1, \mathbf{x}_2) = \left(1 + \frac{w^2 \cdot |\mathbf{x}_1 - \mathbf{x}_2|^2}{\alpha} \right)^{-\alpha}$$

w, λ and α are hyper-parameters, which have to be identified using optimization methods in order to maximize the likelihood function of the multivariate normal distribution. In some cases, the Gamma-exponential correlation function is either exponential ($\gamma=1$) or square exponential ($\gamma=2$). For the Matérn class, the correlation functions with $\nu=3/2$ and $\nu=5/2$ are frequently used.

2.3 Adaptive Gaussian Process

With the variance σ^2 of the multivariate normal distribution, the confidence interval (3σ) of the response surface is available at any point. Thus, it is possible to measure the accuracy of the meta model. Besides the variance, the expected improvement (EI) has been introduced as a second factor for meta model evaluation purposes [4]. EI is defined as a potential improvement which is achieved by investigating the input parameter x :

$$EI = \sigma \left\{ \frac{Y_{\min} - \hat{Y}}{\sigma} \Phi \left(\frac{Y_{\min} - \hat{Y}}{\sigma} \right) + \Psi \left(\frac{Y_{\min} - \hat{Y}}{\sigma} \right) \right\}.$$

$\Phi(\bullet)$ and $\Psi(\bullet)$ are commulative distribution functions and probability density function of the normalized normal distribution. A third evaluation factor is the statistical low bounding (SLB) [5]:

$$SLB = \hat{Y} - k \cdot \sigma \quad \text{with } k = 1, 3, 5, \dots$$

Based on these three factors, the accuracy of the meta model can be improved by using additional support points, which are suggested by the optimization procedures to:

- maximize EI,
- maximize 3σ and
- minimize SLB.

Meta modeling is an adaptive process, which involves several loops of the Gaussian process. Starting from the initial sampling points, the response surface of the modeled system is built. Based on this response surface, additional support points are suggested and make possible that the new response surface is rebuilt more accurately. The process comes to an end either if a predetermined number of support points is computed or if a specified value of maxi-

imum EI is achieved. The necessary number of support points for a specific accuracy of the surrogate model depends on:

- the number of input parameters \mathbf{x} ,
- the correlation between input parameters and
- the degree of nonlinearity of the surrogate model.

The adaptive Gaussian process is very efficient for meta modeling. It requires less support points when compared with other design of experiment (DoE) methods in order to achieve a comparable accuracy.

3 Code-Extraction with OptiY[®]

For system modeling purposes, commercial software systems for CAD/CAE, FEA, CFD, electronic circuit simulation, electro-magnetics, etc. are available. It is better to use tailored programs for the different components of a complex model. The advantages are:

- easy usage and quick handling of the software,
- availability of expert knowledge,
- detailed and accurate component behavior modeling,
- small number of model parameters, which have to be identified.

For the system simulation, different component models created by different software programs have to be coupled. In general, this is difficult to arrange. System models contain component models having a large number of degrees of freedom. This results in a high computing effort for system simulation. Using surrogate models in form of Modelica codes instead of the underlying component models reduces computation time and cost drastically. Such fast system models allow robust design optimization (RDO), e. g. design for six sigma, and reliability based design optimization (RBDO) which both require a large number of runs of system models.

The multitude of software tools normally used for component modeling makes automatic computation of the meta models desirable. The multidisciplinary design software OptiY[®] supports the automatic generation of meta models in form of Modelica code used in system models (Fig. 2) [7]. This software provides generic and direct interfaces to many commercial software systems for CAD, FEA, electronic circuit simulation, CFD, multi body simulation, in-house codes etc. Furthermore, user can easily self-integrate external CAD/CAE-software for ease of use later with a predefined user element and script

template using Visual Basic or C# based on the .NET Framework[®] technology.

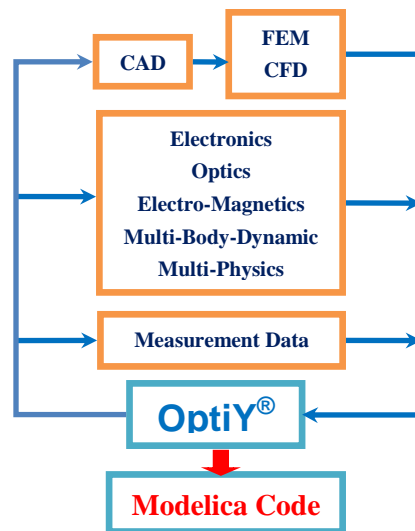


Fig 2: Auto-extraction of Modelica code with OptiY

The numerical algorithms of the adaptive Gaussian process presented in last section are also implemented in OptiY. For this reason an easy and quick-to-use connection to external component models is supported. Combined with the adaptive Gaussian process, this allows the automatic extraction of Modelica codes from external models or data.

With other implemented numerical algorithms in OptiY, following valuable possibilities are also available for the design process:

- Sensitivity Study
- Probabilistic Analysis
- Reliability Analysis
- Robustness Evaluation
- Six Sigma Design
- Robust Design
- Design Optimization
- Data Mining
- Parameter Identification

4 Electromagnetic Actuator

We use a Braille printer with an electromagnetic actuator in order to demonstrate the system simulation with surrogate models [8] (Fig. 3).

In the first step, the system model of the printer consists only of network elements which are taken from the model library of SimulationX [9]. The resulting network schematic is shown in Fig. 4, left side. The

model computes the dynamic behavior of the actuator.

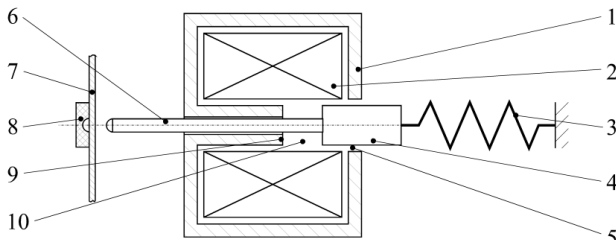


Fig. 3: Braille printer with electromagnetic actuator; 1 back iron, 2 coil, 3 return spring, 4 armature, 5 guiding air gap, 6 needle, 7 paper sheet, 8 die, 9 yoke, 10 working air gap

In the second step, we replace the magnetic network elements by a finite element model applying the software FEMM [10] (Fig 4).

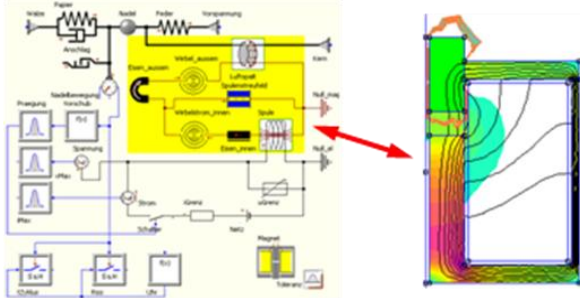
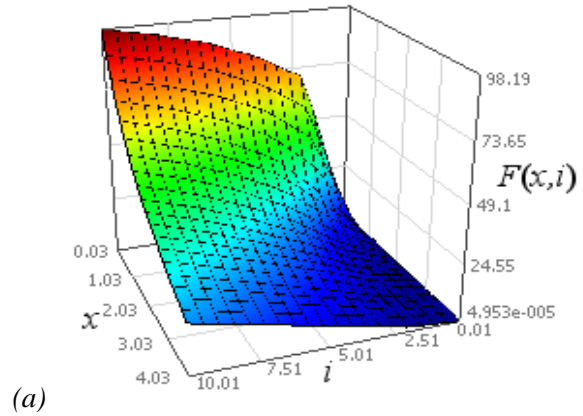


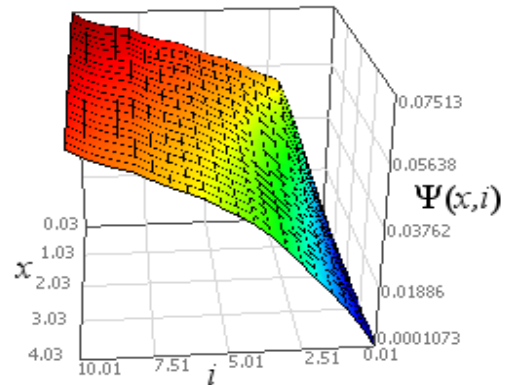
Fig 4: System model of the Braille printer; network model (left) and finite-element model (right) of the magnetic parts of the actuator

OptiY controls the FEMM tool, provides the input vector of the sampling points and collects the simulation results. After the simulation is finished, the surrogate model of the magnetic part of the actuator is computed by the adaptive Gaussian process and exported as Modelica code. All magnetic network elements (yellow block in Fig. 4) are replaced by surrogate Modelica codes of the magnetic force $F_m(x, i)$ and the flux density linkage $\Psi(x, i)$ (Fig. 5). x is the working air gap, i is the coil current. These codes are used to build a new system model in SimulationX.

These modeling steps have to be performed manually and only once. Each update of the surrogate Modelica code can be performed automatically inside of OptiY using the graphical workflow editor.



(a)



(b)

Fig. 5: Surrogate models of the magnetic force $F_m(x, i)$ (a) and the flux linkage $\Psi(x, i)$ (b)

Fig. 6 compares the dynamic behavior of the Braille printer computed with the network model and the surrogate model. The diagram reveals slight differences between the models due to the idealized representation of the coil in the magnetic network element. These differences are particularly notable when the back iron comes into saturation. Therefore, the system model using surrogate Modelica code yields better results of the printer behavior because the underlying finite element model considers spatial inhomogeneity of the magnetic field. However, eddy currents and magnetic hysteresis are neither covered by the network model nor the finite element model. A more detailed description of the example of the Braille printer can be found in [8].

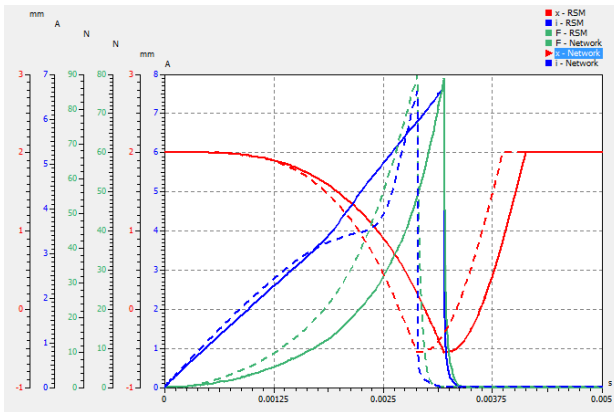


Fig. 6: Dynamic behavior of the Braille printer simulated by a network model (dashed) and a surrogate model (solid); x - printer needle displacement, i - coil current, F - magnetic force

5 Conclusions

Modeling technical systems with network elements is an adequate approach, however, challenging and time-consuming if performed manually. The adaptive Gaussian process is an approach that allows an efficient and automatic generation of precise component models for system modeling. It requires only few support points of the black box system. With additional support points, the accuracy of the computed meta model can be improved step by step if necessary. The mathematical meta model can be written as Modelica code. The algorithms which are needed for this procedure are implemented in the multidisciplinary design software OptiY[®]. It provides generic and direct interfaces to many specialized commercial CAD/CAE-software tools and also in-house codes. Within, user can easily create fast surrogate models and export them as Modelica code automatically.

The study case shows that the meta modeling process is very fast and useful. The amount of identified parameters is smaller in comparison to the network model. The system behavior is more accurate. The application of a Braille printer with an electromagnetic actuator has been demonstrated. Simulation results of a network model and a surrogate model have been compared. The use of fast meta models allows computationally intensive optimization and test procedures, e. g. robust design optimization.

References

- [1] Rasmussen C. E., Williams C. K. I.: Gaussian Process for Machine Learning. MIT Press 2006.
- [2] Santner, T. J., Williams, B. J., Notz, W. I.: The Design and Analysis of Computer Experiment. Springer New York 2003
- [3] Sacks J., Welch W. J., Mitchell T. J., Wynn H. P.: Design and Analysis of Computer Experiments. Statistical Science 4, pp. 409-435, 1989
- [4] Jones, R. D.: A Taxonomy of Global Optimization Methods Based on Response Surfaces. Journal of Global Optimization 21: 345-383, 2001
- [5] Xiong, Y., Chen, W, and Tsui, K.: A New Variable Fidelity Optimization Framework Based on Model Fusion and Objective-Oriented Sequential Sampling. ASME Journal of Mechanical Design , 130 (11), 2008
- [6] Antuolas, A. C.: Approximation of Large-Scale Dynamical Systems. SIAM 2005
- [7] OptiY Software and Documentation. www.optiy.eu
- [8] http://www.optiyummy.de/index.php/Software:_FEM_-_Tutorial_-_Magnetfeld, see Kennfeld-Export als Modelica-Code
- [9] SimulationX Software and Documentation. www.iti.de
- [10] FEMM Software and Documentation. www.femm.info

Modelling of Uncertainties with Modelica

Daniel Bouskela Audrey Jardin
EDF R&D

6 quai Watier, 78401 Chatou Cedex, France
daniel.bouskela@edf.fr audrey.jardin@edf.fr

Zakia Benjelloun-Touimi
IFP Energies nouvelles

1 & 4, avenue de Bois-Préau, F-92852 Rueil-Malmaison Cedex, France
zakia.benjelloun-touimi@ifpenergiesnouvelles.fr

Peter Aronsson

MathCore Engineering
Teknikringen 1F, SE-58330 Linköping, Sweden
peter.aronsson@mathcore.com

Peter Fritzson

Linköping University, Department of Computer and Information Science
SE-58183 Linköping, Sweden
peter.fritzson@liu.com

Abstract

Many industrial applications, e.g. in power systems, need to use uncertain information (e.g. coming from sensors). The influence of uncertain measurements on the behavior of the system must be assessed, for safety reasons for instance. Also, by combining information given by physical models and sensor measurements, the accuracy of the knowledge of the state of the system can be improved, leading to better plant monitoring and maintenance.

Three well established techniques for handling uncertainties using physical models are presented: data reconciliation, propagation of uncertainties and interpolation techniques. Then, the requirements for handling these techniques in Modelica environments are given. They apply to the Modelica language itself: how to specify the uncertainty problem to be solved directly in the Modelica model. They also apply to model processing: what are the pieces of information that must be automatically extracted from the model and provided to the standard algorithms that compute the uncertainties.

Modelica language extensions in terms of two new pre-defined attributes, *uncertain* and *distribution*, are introduced for Real and Integer variables. This is needed to differentiate between *certain* (the usual kind) variables and *uncertain* variables which have associated probability distributions. An algo-

rithm for extracting from the Modelica model the auxiliary conditions needed by the data reconciliation algorithm is given. These new features have been partially implemented in the MathModelica tool (and soon OpenModelica).

Keywords: data reconciliation; propagation of uncertainties; distribution probability laws; Jacobian matrix; Modelica language extensions; model processing

1 Introduction

The major power plant projects at EDF mainly concern improvements of existing plants (e.g., lifetime extension up to 60 years, power upgrading, reliability improvements, etc.) as well as the construction of new plants (e.g., nuclear, renewable energy, etc.). In that context, EDF has acquired a strong background in the modeling and simulation of electrical and power plant applications for improved investigation and operation.

The physical state of a plant is given by sensor measurements which are subject to uncertainties. Hence, good uncertainty assessment is necessary for the proper monitoring of the plant operation set point, and to comply with the safety margins. By combining information given by physical models and sensor measurements the accuracy of the knowledge

of the state of the system can be improved, leading to better plant monitoring and maintenance [6].

Today, engine simulation is important at IFP to solve the problems of air pollution and energy depletion. Many aspects of engine operation are poorly understood because of the problems encountered when attempting to measure the variables that describe the engine's operation. This issue complicates both the engine's control and design. Uncertainty management with parameter sensitivity studies are promising techniques to improve the measurement quality, and subsequently the control and design of internal combustion engines in the years to come.

In order understand the strongly non-linear behaviour of the physical systems under study, IFP has developed a new methodology based on an evolutionary experimental design, kriging and statistical modeling concepts, that are more adequate and accurate than the traditional linear regression techniques [4].

EDF and IFP are currently developing advanced libraries for modeling power plants and engines in Modelica, and are therefore interested in reusing these models for uncertainty computations. The main benefit will lie in the delivery of integrated environments for system modeling and uncertainty studies.

The objective of this article is to make a proposal to extend the Modelica language for the handling of uncertainties. First, several important use cases are presented: data reconciliation, propagation of uncertainties, kriging and response surface methodology. Then the basic requirements for the handling of such techniques with Modelica are established. Finally, a technical proposal for an extension of the Modelica language is given.

2 Techniques for handling uncertainties

2.1 Data reconciliation

The objective of the data reconciliation technique is to improve the knowledge of the physical state of a system using redundant physical measurements of the system and the physical laws that govern the behavior of the system. The increase of knowledge is obtained by reducing the uncertainty intervals of the variables of interest (i.e., that define the state of the system), or in other words by finding the reconciled values of the physical state which are closest to the true values of the physical state, which by definition cannot be exactly known.

By definition, the vector of improvements ν is defined such as:

$$\hat{x} = x + \nu$$

where x and \hat{x} denote resp. the vector of measured values and the vector of improved values, also called reconciled values.

The data reconciliation technique can be formally expressed as an optimization problem, where the goal is to find the vector of improvements such as the objective function ξ_0 attains its minimum value.

$$\xi_0 = \nu^T \cdot S_x^{-1} \cdot \nu \Rightarrow \min_{\nu}(\xi_0)$$

S_x is the covariance matrix, which is symmetric by definition. Its diagonal elements are:

$$S_{x_i}^2 = \left(\frac{w_{x_i}}{\lambda_{95\%}} \right)^2$$

with w_{x_i} being the half-width confidence interval of the measured value x_i , and $\lambda_{95\%} \approx 1.96$ corresponding to a level of confidence of 95%.

Its off-diagonal elements are:

$$S_{x_i,k} = r_{x,ik} \cdot S_{x_i} \cdot S_{x_k}$$

where $r_{x,ik}$ is an empirical (estimated) correlation coefficient. Because of the difficulty of estimating these coefficients, they are often set to zero by making the assumption that the measured variables are uncorrelated. Then:

$$\xi_0 = \sum_i \left(\frac{\hat{x}_i - x_i}{S_{x_i}} \right)^2 = \sum_i \left(\frac{\nu_i}{S_{x_i}} \right)^2$$

The reconciled values are constrained by the physical laws such as mass, energy and momentum balances, state functions, correlations, etc, which are expressed in the mathematical model of the system. The subset of the model equations that constrain the reconciled values are called the auxiliary conditions, and denoted $f(\cdot)$. Hence $f(\hat{x}) = 0$, whereas $f(x) \neq 0$ in general. This is why $f(x)$ is called the vector of contradictions.

The algorithm for computing the reconciled values is given in the VDI 2048 standard [5]. This standard considers that the probability function of each measured value follows a Gaussian distribution law that the measurements are performed while the system is in steady-state, and that the measured values are reliable enough so that the vector of improvements is small.

As a consequence of the two last hypotheses, $f(\cdot)$ needs only to be static, and may be linearized around the measured values:

$$f(x) = -\frac{\partial f(x)}{\partial x} \cdot v = -F \cdot v$$

where F is the Jacobian matrix of the auxiliary conditions.

It can then be shown that the reconciled values are given by:

$$\hat{x} = x - S_x \cdot F^T \cdot (F \cdot S_x \cdot F^T)^{-1} \cdot f(x)$$

The covariance matrix of the reconciled values is given by:

$$S_{\hat{x}} = S_x - S_x \cdot F^T \cdot (F \cdot S_x \cdot F^T)^{-1} \cdot F \cdot S_x$$

The uncertainties associated with the reconciled values can then be computed from the knowledge of this matrix:

$$S_{\hat{x}_i}^2 = \left(\frac{w_{\hat{x}_i}}{\lambda_{95\%}} \right)^2$$

The Gaussian distribution hypothesis is verified a posteriori by applying a χ^2 test on the reconciled values. This check is done globally by checking the following condition:

$$\xi_0 \leq \chi_{r,95\%}^2$$

where $\chi_{r,95\%}^2$ is given by a χ^2 -table and r is the number of auxiliary conditions. Local checks on individual variables may be performed as well.

In practice, the actual implementation of the method differs slightly from its theoretical formulation in order to avoid the cumbersome matrix inversions. To that end, they are replaced by the resolution of linear equations. For instance, the reconciled values are computed using the following procedure:

$$\hat{x} = x - S_x \cdot F^T \cdot f^*$$

with f^* such as:

$$(F \cdot S_x \cdot F^T) \cdot f^* = f(x)$$

f^* may be solved numerically using the Newton algorithm.

2.2 Propagation of uncertainties

From the knowledge of estimated sources of uncertainties, it is possible to derive the uncertainties of variables of interest by propagating the sources of uncertainties through a model of the system.

To that end, EDF R&D has developed a complete methodology. This methodology is divided into four steps, as illustrated in Figure 1.

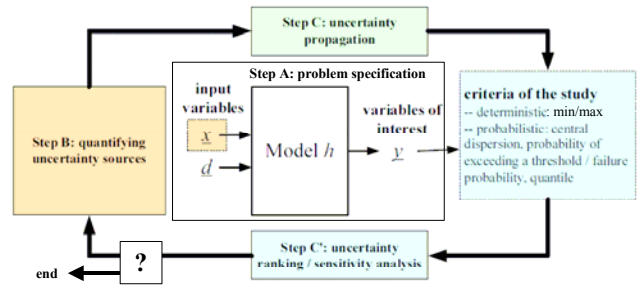


Figure 1: Steps of the uncertainty propagation methodology developed at EDF R&D [8].

Step A aims at identifying the sources of uncertainties \underline{x} , the variables of interest \underline{y} , and the model of the system $h(\cdot)$. The uncertainty study is then formally expressed as:

$$\underline{y} = h(\underline{x}, \underline{d})$$

where \underline{d} denotes the variables treated as certain (i.e. whose uncertainties can be neglected).

A decision criterion must also be defined to indicate how the uncertainties of the variables of interest have to be quantified and to determine the final objective of the uncertainty study. The criterion can be deterministic, by assessing a minimum and a maximum value for each variable of interest, or probabilistic like the probability of exceeding a given threshold, a quantile, or a central dispersion.

Step B aims at quantifying the sources of uncertainties. When the deterministic criterion is chosen, a min and a max value must be associated with each variable. When the probabilistic criterion is chosen, the sources of uncertainties are treated as the components of a random vector \underline{X} . For each individual component X^i , the probability distribution must be assessed. Also, the statistical dependency between two components X^i and X^j should be evaluated in the form of correlation coefficients.

Step C is the uncertainty propagation through the model. When a deterministic criterion is chosen, finding the minimum and the maximum values of \underline{y} is quite easy if the model is monotonous wrt. \underline{x} . Otherwise this search may become a potentially complex optimization problem. To alleviate this difficulty, some optimization algorithms or simplified approaches based on design of experiments to estimate extreme values of \underline{y} may be used. When a probabilistic criterion is chosen, the difficulty is to characterize the probability distributions of the random vector $\underline{Y} = h(\underline{X}, \underline{d})$. For the assessment of expectation/variance or the probability of exceeding a threshold, both approximation methods (e.g. quadratic combination, FORM-SORM methods) and sam-

pling methods (e.g. Monte Carlo simulations, Latin Hypercube simulations) can be used. For the assessment of a quantile, only sampling methods can be applied (e.g. Wilk's formula). Contrary to approximation methods, sampling methods make no assumption on the model $h(\cdot)$, but may be very CPU-time consuming.

Step C' is the ranking of the sources of uncertainties and the sensitivity analysis. Chosen indicators are used to rank the uncertainty sources with respect to their impact on the uncertainties of the system's characteristics of interest. Depending on the result of this ranking, the modeling of the sources of uncertainty can be adjusted (or some sources of uncertainties can be neglected) to perform another propagation.

An uncertainty study rarely finishes after a first round of steps A, B, C and C'. Step C' actually plays a crucial role since ranking the results highlights the variables that truly determine the relevance of the final results. If the uncertainty laws of some input variables have been chosen too roughly during step B, it is necessary to collect additional information on the influential sources of uncertainty and re-apply the whole methodology to refine the analysis, and so-on until satisfaction.

Probability distributions may be expressed in the form of parametric distribution laws with the help of a limited number of parameters $\underline{\theta}$.

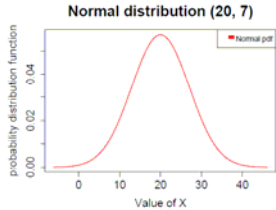
Two examples of such parametric distribution laws are given below [8].

Normal (or Gaussian) distribution

$\underline{\theta} = (\mu, \sigma)$

μ is the mean value

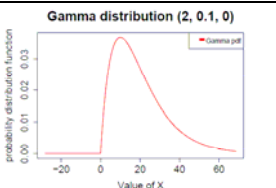
σ is the standard deviation

$$f_X(x, \underline{\theta}) = \frac{1}{\sigma \cdot \sqrt{2 \cdot \pi}} \cdot \exp\left(-\frac{1}{2} \cdot \left(\frac{x - \mu}{\sigma}\right)^2\right)$$


Gamma distribution

$\underline{\theta} = (\lambda, k, \gamma)$

$\lambda > 0, k > 0$



$$f_X(x, \underline{\theta}) = \frac{\lambda}{\Gamma(k)} \cdot (\lambda \cdot (x - \gamma))^{k-1} \cdot \exp(-\lambda \cdot (x - \gamma)) \cdot 1_{\gamma}(x)$$

where $\Gamma(\cdot)$ is the gamma function

$$\Gamma(k) = \int_0^{+\infty} t^{k-1} \cdot \exp(-t) \cdot dt$$

The choice of the correct distribution law depends on the application. For instance, the normal distribution is relevant in metrology.

The dependency between variables may be expressed using the copula theory or the Pearson correlation coefficient [15, 16]. The latter is defined as:

$$\rho_{x^i, x^j}^p = \frac{Cov(X^i, X^j)}{\sigma_{X^i} \cdot \sigma_{X^j}}$$

where $Cov(X^i, X^j)$ is the covariance between X^i and X^j , and σ_{X^i} and σ_{X^j} are respectively the standard deviations of X^i and X^j .

Several techniques may be used for propagating the uncertainties.

The quadratic combination method is a probabilistic approach based on the Taylor decomposition of \underline{Y} wrt. \underline{X} around the mean point $\underline{\mu}_X$.

The Monte-Carlo method is a numerical integration method using sampling, which can be used, for example, to determine the expectation μ_{Y^i} and the standard deviation σ_{Y^i} of each variable of interest Y^i .

Several ranking techniques may be used, such as those based on the quadratic combination's importance factor or the Pearson correlation. They aim at finding the influence of the inputs X^i on the outputs Y^j .

2.3 Interpolation techniques

Mathematical models of physical systems are important tools in many fields of scientific research. But better knowledge of systems behavior and increasingly desired accuracy lead to higher complexity of models, which, in this context, sometimes are not sufficient to meet the expectations of the experimenters.

Uncertainty studies must in particular be adapted to handle such complex models. The following section describes some examples of advanced methods that are especially used at IFP: the kriging method, the experimental design theory and the Response Surface Methodology (RSM).

2.3.1 Kriging with non-linear trend

One of the studies at IFP using the kriging method is the analysis of a catalytic system for pollution control, which consists in post-treating smoke produced by diesel engines through NOx trap (i.e. Nitrogen-Oxides trap [3]).

As a surrogate of the real system, a kinetic model was developed to represent the physico-chemical phenomenon, depending on parameters (e.g. pre-exponential factors, activation energies, adsorption constants) that cannot be obtained from theoretical considerations. Therefore, experiments are required to calibrate the model. A criterion is suggested for experimental designs adapted to kinetic parameters identification, when the model is highly non-linear and the kinetic model does not fit well experimental data. These differences observed between kinetic model and experimental data, can be represented by a Gaussian process realization. Gaussian process often accounts for correlated errors due to lack of fit.

More explicitly, the model is represented by:

$$y = f(x, \beta)$$

where y is the response vector (e.g. the NOx concentration), x the experimental conditions and β the kinetic parameters of the model represented by the non-linear function f .

Then, the first model is corrected and replaced by:

$$y = f(x, \beta) + Z_{\sigma^2, \theta}(x)$$

where $Z_{\sigma^2, \theta}(x)$ is a centered Gaussian process with Gaussian covariance kernel specified by a variance σ^2 and a vector θ of scale parameters.

As an example, the covariance kernel can be given by a kriging approach which is commonly used in the field of computer experiments. However, in traditional use, the trend is linear. Its estimation is obtained through an analytical formula as well as its uncertainty.

The first difficulty is to estimate the trend parameters considering its non-linear behavior. Similarly to non-linear regression, the traditional analytical formula for β is then replaced by a minimization procedure. In this case, the theory of kriging with non-linear trend can be applied as summarized below.

The covariance kernel of the centered Gaussian process is defined by:

$$\text{Cov}(Z_{\sigma^2, \theta}(x), Z_{\sigma^2, \theta}(x+h)) = \sigma^2 \cdot R_{\theta}(h)$$

where the Gaussian spatial correlation is used and expressed by:

$$\forall h \in \mathfrak{R}^k, R_{\theta}(h) = \exp\left(-\sum_{i=1}^k \theta_i h_i^2\right)$$

Let m be the number of design points and $y = (y_1 \dots y_m)^T$ the outputs observed at location $s = (s_1 \dots s_m)^T$, $s_i \in \mathfrak{R}$. Using maximum likelihood estimation, expression of kriging predictor \hat{y} and variance prediction φ at a new location x_0 are given by:

$$\hat{y}(x_0) = r \cdot R^{-1} \cdot y - (F^T \cdot R^{-1} \cdot r - f)^T \cdot (F^T \cdot R^{-1} \cdot F)^{-1} \cdot F^T \cdot R^{-1} \cdot y$$

$$\varphi(x_0) = \sigma^2 \cdot \left(1 + \|F^T \cdot R^{-1} \cdot r - f\|_{F^T \cdot R^{-1} \cdot F} + \|r\|_R\right)$$

where:

$$\|u\|_A = u^T \cdot A^{-1} \cdot u$$

$$r = (R_{\hat{\theta}}(x_0 - s_1) \dots R_{\hat{\theta}}(x_0 - s_m))^T$$

$$\forall i \in [1; m], \forall j \in [1; m], R_{ij} = R_{\hat{\theta}}(s_i - s_j)$$

$$F = f(s, \hat{\beta})$$

$$f = f(x_0, \hat{\beta})$$

and the parameters are obtained by solving recursively and simultaneously the following simultaneous equations:

$$\hat{\beta} = \min_{\beta} \left((y - F)^T \cdot R^{-1} \cdot (y - F) \right)$$

$$\hat{\sigma} = m^{-1} \cdot (y - F)^T \cdot R^{-1} \cdot (y - F)$$

$$\hat{\theta} = \arg \min \left(\hat{\sigma}^2 \cdot |R^{-1}|^{1/m} \right)$$

The minimization algorithm determines $\hat{\theta}$ through the modified Hooke and Jeeves method, described in Kowalik and Osborne [11]. For more details, see Lophaven *et al.* [12]

Notice that the predictor \hat{y} and the variance prediction φ depend on the estimator β through F and f . As a consequence, in universal kriging, kriging predictor and prediction variance expressions cannot be interpreted as conditional expectation and variance (see Helbert *et al.* [13]). This is due to the fact that only uncertainties induced by the estimation of trend parameters are taken into account, and not those created by approximating variance and correlation parameters. Hence, it can underestimate the uncertainty on the response and lead to very important difficulties for non-linear models.

2.3.2 Experimental Design theory and Response Surface Methodology

The Response Surface Methodology (RSM) has been described in detail by Dejean *et al.* [14]. The purpose

is to approximate a complex process with respect to uncertain parameters belonging to a region of interest. Engineers define a performance measure of the process called response y (e.g. cumulative oil production, field net present value, ...) and some input variables x_1, x_2, x_3, \dots called factors, that are assumed to influence the response (e.g. petrophysics, field structural map, well locations, economic factors, ...). The input variables correspond to the prior uncertainties on the process. RSM provides tools for identifying the factors that are influential factors, such as illustrated below:

$$y = a_0 + a_1 \cdot x_1 + a_2 \cdot x_2 + a_3 \cdot x_3 + \dots + a_1 \cdot a_2 \cdot x_1 \cdot x_2 + \dots$$

where a_0, a_1, \dots are constant coefficients obtained by fitting a set of numerical simulations.

The main interest of this RSM model is its negligible cost to get new values of the response compared to CPU-time consuming simulations. This regression model can then be used to make predictions of the process over the uncertain domain and to generate probabilistic distribution of the response using Monte Carlo sampling technique.

A sufficient number of response values corresponding to different factor values is necessary in order to fit this model. These values should be representative enough of the behavior of the response in the domain of variation of the factors. Thus the experimental design theory is applied, since, for a given objective and a given uncertain domain, it delivers the right set of model simulations to be performed in order to properly model the response behavior in the uncertain domain. Many experimental designs are available depending on both the objective (sensitivity study or risk analysis study) and the acceptable CPU-time. The selected model simulations must:

- (a) be numerous enough so that all the coefficients a_i of the model can be estimated and
- (b) ensure good quality of the model, both in terms of accuracy and prediction.

For sensitivity studies, the objective is to identify the uncertain parameters that influence the response. In that case, the RSM model does not need to be very accurate and classical sensitivity designs are the two-level fractionals.

For risk analysis purposes, the RSM model should be of good quality in order to deliver accurate predictions. The composite designs are in that case the most appropriate. All those designs are well known and tabulated. They have the best properties with respect to the objective of the study, but they can still be too expensive or on the contrary too coarse in a

specific context. In that case, some other designs such as small composite designs or optimal designs can be used.

Once the RSM model has been fitted, it can be used to compute probabilistic distributions of the production forecasts as a function of the main model uncertainties. This is commonly called a technical risk analysis.

3 Requirements on the Modelica language and tools

In this section, the requirements are given independently from the existing Modelica specifications and tools. There are two kinds of requirements: those who apply to the language itself, and those who apply to the model processor. The model processor is defined as the tool that has the analysis and symbolic manipulation capabilities to produce the desired result from the Modelica model.

3.1 Identifying the uncertain variables

In the following paragraphs, only continuous variables will be considered.

There are two broad kinds of variables: *certain* or *uncertain*. *Certain* variables are single valued variables, whose values are known (explicitly given as inputs of the model), or unknown (implicitly given as outputs of the model). *Uncertain* variables are random variables which represent probability distributions.

Formally, a certain variable could be seen as an uncertain variable with a normal distribution of zero standard deviation (i.e. a Dirac function). It is however preferable to continue handling these two types of variable separately, because of the infinities associated with Dirac functions.

In the case of uncertainty propagation, the sources of uncertainty are the inputs and the variables of interest are the outputs of the computation. Input means that the distribution law of the variable is given, and output means that the distribution law is computed. Output uncertain variables may also be called observation variables.

In the case of data reconciliation, uncertain variables are considered as both inputs and outputs, the difference between the outputs and the inputs being the vector of improvements.

Note that the words “input” and “output” should not be confused here with the semantics of the Modelica keywords “input” and “output”. As they indeed have

different meanings, other keywords will be used when implementing these notions in Modelica, as described in Section 4.

[R1] The Modelica language should give the possibility to declare uncertain variables as inputs only or as outputs only (for the propagation of uncertainties), or as inputs/outputs (for data reconciliation).

3.2 Assigning parametric distribution laws to uncertain variables

An uncertain variable x_i may be characterized by the parameters of its distribution law $\underline{\theta}_{x_i} = (\alpha_{x_i}, \beta_{x_i}, \dots)$.

In the frequent case of a normal distribution, these parameters are the mean value μ_{x_i} and the standard deviation σ_{x_i} . Then, x_i may be written as:

$$x_i = \mu_{x_i} \pm w_{x_i}$$

where the half-interval w_{x_i} is a multiple of σ_{x_i} :

$$w_{x_i} = \lambda \cdot \sigma_{x_i}$$

λ being a function of the level of confidence. For instance, for a level of confidence of 95%, $\lambda_{95\%} = 1.96$. This expression means that there are 95% chances that the value of x_i is in the interval $[\mu_{x_i} - \lambda_{95\%} \cdot \sigma_{x_i}; \mu_{x_i} + \lambda_{95\%} \cdot \sigma_{x_i}]$.

[R2] The Modelica language should give the possibility to assign one parametric distribution law to each uncertain variable by specifying its name (e.g. normal distribution) and the values of its parameters (e.g. mean value and standard deviation, or alternatively mean value, confidence level and half-interval in the case of a normal distribution).

3.3 Specifying dependencies between uncertain variables

[R3] The Modelica language should give the possibility to specify the dependencies between uncertain variables in the form of correlation matrices (covariance matrices, matrices of Pearson correlation coefficients, etc.).

3.4 Handling redundant information for data reconciliation

When performing data reconciliation, redundant information is fed into the model as shown in the following example (see Figure 2).

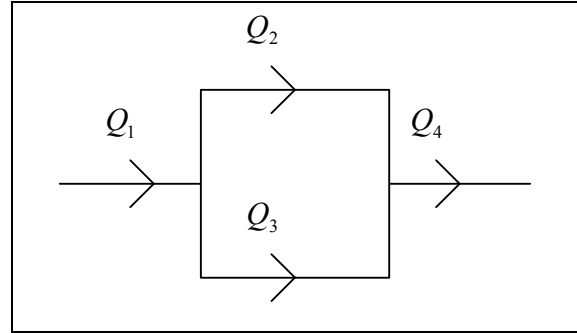


Figure 2: Example of data reconciliation for flow measurements

In Figure 2, the hydraulic circuit is instrumented with four flow meters, which give respectively the following values and uncertainties (all quantities are given in kg/s):

$$\begin{cases} Q_1 = 5.0 \pm 1.0 \\ Q_2 = 2.5 \pm 0.5 \\ Q_3 = 2.6 \pm 0.1 \\ Q_4 = 5.5 \pm 0.5 \end{cases}$$

Writing the mass balance equations for the system yields:

$$\begin{cases} Q_1 = Q_2 + Q_3 \\ Q_4 = Q_2 + Q_3 \end{cases}$$

Considering the values at the center of the uncertainty intervals, it is obvious that the equations system is not satisfied. However, the equations may be satisfied by carefully choosing the proper values within the uncertainty ranges. This is what is actually done when applying the data reconciliation technique, which yields:

$$\begin{cases} \hat{Q}_1 = 5.3 \pm 0.3 \\ \hat{Q}_2 = 2.7 \pm 0.3 \\ \hat{Q}_3 = 2.6 \pm 0.1 \\ \hat{Q}_4 = 5.3 \pm 0.3 \end{cases}$$

Note that the uncertainty intervals have been reduced.

At the present time, the Modelica language only handles square systems of physical equations (having as many unknowns as equations). The question is how to consider the four variables for the model: are they inputs or outputs? If they are all considered as inputs, then the system is over-constrained. If they are all considered as outputs, then the system is under-constrained. Two of them could be considered as inputs, and the other two as outputs (6 possibilities) to obtain a square system. Note that the last alterna-

tive is applicable to the reconciled values, all possibilities being equivalent as the reconciled values satisfy the auxiliary conditions.

The most natural way is to consider the four variables as both inputs and outputs of the data reconciliation algorithm, and use requirement [R1] to that end, so no additional requirement on the Modelica language is necessary. However, as the term “input/output” for data reconciliation is somewhat ambiguous, because it may be confused with the terms “input” and “output” for DAE simulation, the vector of inputs/outputs for data reconciliation will be called the vector of control variables (or control vector) in the rest of the paper, because it controls (defines) the state of the system.

Once the data reconciliation algorithm is completed and the variables are assigned their reconciled values, they can be considered as standard inputs or outputs to have a square system, as required for DAE simulation. Doing so, the same model could be used for data reconciliation and simulation. Data reconciliation would be used to compute an improved state from redundant measurements, and the result would be readily used as the initial state of subsequent simulations.

In the above example, all mass flow rates would be declared as inputs/outputs for data reconciliation, but only two of them (no matter which) would be declared as standard input variables for simulation.

3.5 Extracting the auxiliary conditions for data reconciliation

The auxiliary conditions $f(\cdot)$ constitute the subset of the model equations that constrain the control variables. $f(\cdot)$ must be extracted from the model because, as shown in Section 2.1, $f(x) \neq 0$ before data reconciliation and $f(x)$ must be evaluated by the data reconciliation algorithm, where x is the control vector.

The extraction of $f(\cdot)$ should be fully automatic, as the auxiliary conditions may be scattered throughout the whole model, which is usually a graph of connected model components.

[R5] The model processor should be able to extract the auxiliary conditions from the model.

3.6 Computing the Jacobian matrix of the auxiliary conditions for data reconciliation

The Jacobian matrix of the auxiliary conditions is defined as:

$$F = \frac{\partial f(x)}{\partial x}$$

where $f(\cdot)$ are the auxiliary conditions, extracted from the model equations, and x is the control vector. Note that F is not square as the number of control variables is greater than the number of auxiliary conditions due to the fact that data reconciliation is based on the use of redundant information.

[R6] The model processor should be able to compute the Jacobian matrix of the auxiliary conditions wrt. the control vector.

3.7 Performing the data reconciliation algorithm

Once the quantities x , S_x , $f(\cdot)$ and F are known, it is possible to run the data reconciliation algorithm. Two alternatives are possible. The first is to write a Modelica script of the algorithm that could be compiled with the model. The second is to write a program in another environment such as Python [7], that would have access to a Modelica functional interface that would provide the values of the above quantities upon request from the main program.

In the above example:

$$x = (Q_1 \quad Q_2 \quad Q_3 \quad Q_4)^T$$

$$S_x = \text{diag} \left(\left(\frac{w_{Q_1}}{\lambda_{95\%}} \right)^2, \left(\frac{w_{Q_2}}{\lambda_{95\%}} \right)^2, \left(\frac{w_{Q_3}}{\lambda_{95\%}} \right)^2, \left(\frac{w_{Q_4}}{\lambda_{95\%}} \right)^2 \right)$$

$$f(x) = \begin{pmatrix} Q_1 - Q_2 - Q_3 \\ Q_2 + Q_3 - Q_4 \end{pmatrix}$$

$$F = \begin{pmatrix} 1 & -1 & -1 & 0 \\ 0 & 1 & 1 & -1 \end{pmatrix}$$

[R7] The model processor should be able to provide x , S_x , $f(\cdot)$ and F to the data reconciliation algorithm.

3.8 Performing uncertainty propagation algorithms

The uncertainty propagation methodology is implemented in OpenTURNS, which is an open source tool developed by EDF/R&D, Phimeca and EADS [8].

The knowledge of the sources x , the model $f(\cdot)$ and the variables of interest y is sufficient to run the OpenTURNS algorithms. $y = f(x)$ must be evalu-

ated upon request from OpenTURNS, and its value y returned to OpenTURNS.

Note that it is much easier to use OpenTURNS than the data reconciliation in a Modelica environment, as in the case of OpenTURNS no processing is required on the model other than identifying which variables constitute the vectors x and y . $f(\cdot)$ is the whole original model, as compared to the extracted auxiliary conditions for data reconciliation.

The uncertainty propagation algorithms may thus be performed by coupling the Modelica environment with OpenTURNS, as shown in Figure 3. The coupling principle is to automatically generate the interface between OpenTURNS and the simulation code compiled from the Modelica model. This can especially be done by extending the Modelica compiler.

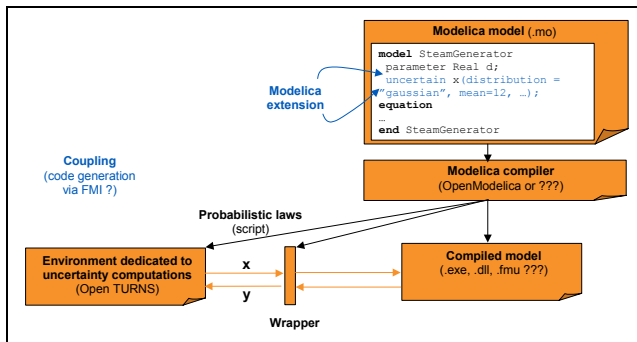


Figure 3: Principle for coupling OpenTURNS with a Modelica platform

[R8] The model processor should be able to interface the model with OpenTURNS.

4 Proposal for a Modelica language extension

This section gives a preliminary design proposal of Modelica language extensions for supporting uncertainties. This proposal will be considered for test implementation in MathModelica [10] and OpenModelica [9].

The proposal to support requirement [R1], described in Section 3.1, is to introduce a *new attribute* for the built-in classes Real and Integer. This attribute should have an enumeration type that allows specifying it as given (i.e., kind of “input”), sought (kind of “output”) or refine (kind of “input/output”) whose uncertainty is refined). In order to distinguish it from the already established semantics for input and output variables in Modelica we propose the following naming:

```

type Uncertainty = enumeration(
  given "a known uncertainty",
  sought "an unknown uncertainty",
  refine "a known uncertainty to be refined"
);
    
```

The attribute is named `uncertain`, here showed for the Real type (as described in the Modelica language specification):

```

type Real // Note: defined with Modelica syntax although predefined
...
parameter Uncertainty uncertain;
equation
...
end Real;
    
```

Let us illustrate how this is used by an example:

```

model Process
parameter Real p1=0.1;
Real v1;
Real v2(uncertain=Uncertainty.given);
equation
...
end Process;
    
```

To support [R2], described in Section 3.2, we propose another *new attribute* called `distribution` for the built-in classes Real and Integer.

```

type Real // Note: defined with Modelica syntax although predefined
...
parameter Distribution distribution;
equation
...
end Real;
    
```

The following distributions are proposed (tool vendors could be allowed to extend this list themselves):

```

partial record Distribution
parameter String name;
end Distribution;

record NormalDistribution
extends Distribution(
final name="Normal");
parameter Real mu,sigma;
end NormalDistribution;

record ExponentialDistribution
extends Distribution(
final name="Exponential");
parameter Real lambda,gamma;
end ExponentialDistribution;
    
```

```

record GammaDistribution
  extends Distribution(final name="Gamma");
  parameter Real lambda,k,gamma;
end GammaDistribution;

record LogNormalDistribution
  extends Distribution(
    final name="LogNormal");
  parameter Real mu, sigma,gamma;
end LogNormalDistribution;

record TriangularDistribution
  extends Distribution(
    final name="Triangular");
  parameter Real a,b,m;
end TriangularDistribution;

record UniformDistribution
  extends Distribution(
    final name="Uniform");
  parameter Real a,b;
end UniformDistribution;

record BetaDistribution
  extends Distribution(
    final name="Beta");
  parameter Real r,t,a,b;
end BetaDistribution;

record PoissonDistribution
  extends Distribution(
    final name="Poisson");
  parameter Real mu;
  end PoissonDistribution;

```

The rationale for introducing the distributions and the uncertainty properties as attributes for the built-in classes Real and Integer is the flexibility that they bring. With this approach it becomes possible, for instance, to change several uncertainties at once by using parameterization:

```

model Process
  replaceable
  parameter Distribution d =
    Distribution("");
  parameter Uncertainty u =
    Uncertainty.given;
  Real x(distribution=d, uncertain=u);
  Real y(distribution=d, uncertain=u);
  ...
end Process;

```

The requirement [R3] to support the addition of dependencies between variables is performed at the top

level of a model. It is only there that the modeler knows what the dependencies are, since they might appear due to e.g. connections of components. What is required is to be able to express a dependency between two uncertain variables and give a correlation coefficient. This coefficient can be of different kinds, as briefly mentioned in Section 3.3. The user has to select one kind of coefficients, e.g. covariance or Pearson. It is not possible to mix different kinds in a model. We propose to support R3, not by extending the Modelica language, but instead by allowing the user to introduce a set of equations at the top level that the tool can recognize, as follows:

```

model System
  Process p1,p3;
  Process2 p2;
  CovCorrelation[:] covCorrelation =
    {CovCorrelation(p1.q,p2.q,0.1),
     CovCorrelation(p2.q,p3.q,0.2)};
end System;

```

The rationale for this design to support R3 is that there is no need for a language extension (which avoids to make the language more complex). A similar record definition is required for Pearson correlation coefficients.

With the above language extensions, it becomes possible for a Modelica tool to automatically fulfill requirements [R4, R5 and R6]. How this is performed is explained in the next section.

5 New features for model processing

Apart from the necessary language extensions proposed in previous section, the analysis of models with uncertainties requires some new features from a Modelica tool perspective.

For data reconciliation it is required that the set of equations f that constrain the control variables can be extracted. Remember that these equations together with the uncertain variables typically result in an underdetermined system, i.e., there are more variables than equations. The method of extracting the equations is as follows. Given a model M with variables vector X :

1. Perform a causality analysis by running BLT sorting on the complete system. This yields a series of blocks B_i ($i=1$ to n).
2. Remove from M all blocks B_i that are square wrt. the subset of X solved in earlier blocks ($B_1 \dots B_{i-1}$) referenced by B_i .

3. Remove from M all blocks B_i that do not influence X (i.e., downstream the causality analysis from X).
4. If there are one or more components from X that do not belong to any of the remaining blocks B_i , raise an error.
5. Otherwise, f is the set of the equations corresponding to the remaining blocks B_i .

The result from step 5 is the set of equations that constrain the control variables. This approach has one problem that needs some attention. The blocks identified can both contain uncertain control variables and normal variables. In such a case, it can be necessary to eliminate the normal variables to reduce the equations to only contain control variables. Such eliminations can only be performed if for instance all inverses of used functions are available.

The tool will also construct the Jacobian matrix of this function as required by the data reconciliation algorithm.

The connection with OpenTURNS to be able to perform uncertainty propagation is straightforward. OpenTURNS simply requires a computational block that can compute the outputs given a certain input.

6 Results

The proposed language extensions have been implemented in the OpenModelica compiler frontend, and the extraction algorithm for data reconciliation presented in the previous section has been implemented in MathModelica.

The example in Figure 4 is used as a test case. It is the model of a fluid pipe system with a pump feeding the system and a volume collecting the output flow from the system.

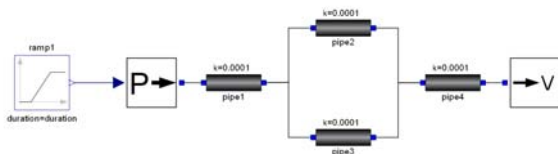


Figure 4 Example of pipe system with uncertainties

The pipe model is defined as:

```

model Pipe
  import SI = Modelica.SIunits;
  Port port_a;
  Port port_b;
  SI.VolumeFlowRate q
    "flow from port_a to port_b";
  SI.Pressure dp "pressure drop over pipe";

```

```

parameter Real k=0.0001
  "Friction factor";
parameter Real rho=1000.0
  "density (water by default)";
equation
  q=port_a.q;
  dp=port_a.p - port_b.p;
  port_a.q + port_b.q=0;
  dp=k/rho*q*abs(q);
end Pipe;

```

The uncertain variables are declared at the system level as follows (all pipes have q as uncertain variable to be refined):

```

model PipeSystem
  Pipe pipe1(q.uncertain = Uncertainty.refine);
  Pipe pipe2(q.uncertain = Uncertainty.refine);
  ...
end PipeSystem;

```

Figure 5 shows a simulation of the pipe system when a ramp signal is applied to the pump. Due to the same friction coefficients of the pipe segments the flows are symmetrically distributed through the system.

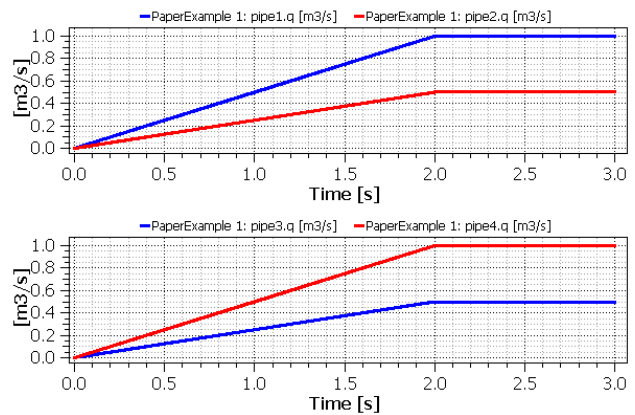


Figure 5 Simulation of the pipe system with MathModelica

Running the extraction algorithm and thereafter eliminating undesired variables result in the following equations, which constitute $f(X)$ (note that X here is $\{q_1, q_2, q_3, q_4\}$):

$$\begin{aligned}
 q_1 - q_2 - q_3 &= 0; & (1) \\
 0.0001 * q_2 * \text{abs}(q_2) - 0.0001 * q_3 * \text{abs}(q_3) &= 0; & (2) \\
 q_2 + q_3 - q_4 &= 0; & (3)
 \end{aligned}$$

In this example all undesired variables have been eliminated resulting in a system only containing the

uncertain variables that are candidates for refinement. Equations 1 and 3 originate from the connect statements (flows are summed up to zero), and equation 2 comes from the pressure drop over the pipes. This example is by intent very similar to the example in Figure 2, so we can reuse the same measured values and uncertainty intervals. The only difference is that we have here three constraint equations instead of two.

The data reconciliation algorithm has been run with Mathematica, that used the auxiliary equations extracted by MathModelica from the original model. Below are the calculations for new measurements and confidence matrix:

```

xr=x-Sx.Transpose[F].Inverse[F.Sx.Transpose[F]].fx
{{5.22801},{2.61526},{2.61275},{5.22801}}
Sxr=Sx-Sx.Transpose[F].Inverse[F.Sx.Transpose[F]].F.Sx
{{0.00865708,0.00441341,0.00424367,0.00865708},
{0.00441341,0.00224998,0.00216344,0.00441341},
{0.00424367,0.00216344,0.00208023,0.00424367},
{0.00865708,0.00441341,0.00424367,0.00865708}}
    
```

This results in new estimates as:

- Q1 = 5.2 ± 0.2
- Q2 = 2.6 ± 0.1
- Q3 = 2.6 ± 0.1
- Q4 = 5.2 ± 0.2

This result differs somewhat from the original example in Figure 2. The reason for this is the extra equation for the pressure drop, which adds another constraint to the system. A reasonable conclusion is that this reconciliation run performs a better job compared to the original example, simply because it has more knowledge of the system in the form of one additional constraint.

7 Conclusions

Two techniques for the handling of uncertainties with Modelica have been presented, and the requirements from a modeling language and tool perspective have been identified. Furthermore their support in the OpenModelica and MathModelica tools has been implemented in order to be able to specify the uncertainties directly in the Modelica models and to extract automatically the necessary pieces of information for the data reconciliation algorithms. The extraction algorithm has been tested and verified on a simple Modelica model. The results seem promising for future developments and real industrial validation, which will be done in the near future in the

scope of the OPENPROD project. The connection with OpenTURNS has not been implemented yet, but the authors foresee no major issues in this work, which is also planned to be done soon.

Acknowledgements

This work was partially supported by the pan-European ITEA2 program and the French and Swedish governments through the OPENPROD project.

References

- [1] Fang K.T., Li R., Sudjianto A., *Design and Modeling for computer experiments*, Chapman & Hall/CRC, 2006
- [2] Santner T., Williams B., Notz W., *The Design and Analysis of Computer Experiments*, Springer, Berlin, 2003.
- [3] Canaud M., Wahl F., Helbert C., Carraro L., *Design of experiments for smoke depollution from the output of diesel engine*, submitted.
- [4] Ferraille M., Busby D., *Uncertainty management on a reservoir workflow*, in International Petroleum Technology Conference, IPTC13768, 2009.
- [5] Verein Deutscher Ingenieure, *Uncertainties of measurement during acceptance tests on energy-conversion and power plants – Fundamentals*, Standard VDI 2048, 2000.
- [6] Zornoza J., Favennec J.-M., Szaleniec S., Piedfer O., *Feedwater Flow-rate And Thermal Power Monitoring And Adjustment By Data Reconciliation In NPPs*, in Proceedings of the 51st ISA POWID Symposium, June 8-13, Scottsdale, USA, 2008.
- [7] Torabzadeh-Tari M., Fritzon P., Sjölund M., Pop A., *OpenModelica-Python Interoperability Applied to Monte Carlo Simulation*, in Proceedings of the 50th Scandinavian Conference on Simulation and Modeling (SIMS'2009), available at www.scan-sims.org. Fredericia, Denmark. October 7-8, 2009.
- [8] EDF-EADS-PhiMeca, *OpenTURNS (version 0.13.1) – Reference Guide*, 2007, available at: <http://trac.openturns.org/wiki/Documentation>.
- [9] Open Source Modelica Consortium. OpenModelica, www.openmodelica.org
- [10] MathCore Engineering AB. MathModelica. www.mathcore.com

- [11] Kowalik J., Osborne M.R., *Methods for unconstrained optimization problems*, Elsevier, New York, USA, 1968.
- [12] Lophaven *et al.*, *Aspects of the Matlab toolbox DACE*, in Informatics and mathematical modeling, DTU, 44p., 2002.
- [13] Helbert *et al.*, *Assessment of uncertainty in computer experiments: from universal kriging to Bayesian kriging*, in Applied Stochastic Models in Business and Industry, vol. 25, pp. 99-113, 2009.
- [14] Dupraz P., *Modélisation et commande avancées d'un moteur diesel à injection directe*, 1998.
- [15] Dixon W.J., Massey F.J., *Introduction to statistical analysis (4th edition)*, McGraw-Hill, 1983.
- [16] Embrechts P., Lindskog F., McNeil A., *Modeling Dependence with Copulas and Applications to Risk Management*, in Handbook of Heavy Tailed Distributions in Finance, ed. S. Rachev, Elsevier, Chapter 8, pp. 329-384, 2003.

Recording of Model Frequency Responses and Describing Functions in Modelica

Tilman Bunte
 German Aerospace Center (DLR)
 Institute of Robotics and Mechatronics, Germany
Tilman.Buente@dlr.de

Abstract

An assistant Modelica package is introduced which supports the determination of model frequency responses or describing functions of Modelica models, as the case may be. The result is frequency response data which can be used for further analysis such as stability properties of the system in closed loop control or the derivation of linear time invariant (LTI) model approximations. The paper addresses inter alia proper scheduling of excitation frequency and amplitude, a brief theory of describing functions (harmonic linearization), the Modelica classes implemented in the package, and some application examples.

Keywords: Frequency response; describing function; system identification.

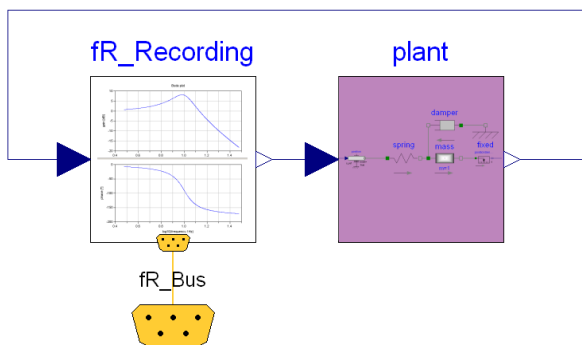


Figure 1: Total Modelica model for frequency response data acquisition of a plant model.

1 Introduction

Frequency responses are applied both for analysis and design of control systems. Multiple useful graphical representations of the frequency response exist like the Bode plot, the frequency response locus (also denoted Nyquist plot for open loop considera-

tions), or the Nichols chart. A number of established criteria for evaluating the dynamic properties of linear systems refer to their frequency response. Also *open loop shaping* and *closed loop shaping* control design methods are based on frequency responses. Moreover, some methods for system identification are available which are based on frequency response data.

1.1 Approaches for frequency response determination

Two methods for frequency response determination exist which are fundamentally different. The first method is to develop a white box model based on physical insight. After operations such as linearization and further model transformation there is a Laplace transfer function which may be evaluated for any value of the frequency ω along the imaginary axis $s = j\omega$. With Modelica models, the first approach is already half way done; the Bode plot of the linearized model can be used for comparison and assessment of the second one.

The second approach is a black box approach applying correlation methods to experimentally acquired in- and output data. The focus of this paper is on the second method whereupon “experiment” is considered the simulation of a Modelica model. The tool provided with the presented package yields frequency response data sampled over frequency being well comparable with *FRD models* in Matlab.

1.2 Frequency response vs. model linearization

Linearization of Modelica models may not be suitable to obtain the aimed result in all cases. Modelica modeling and simulation tools like Dymola or MapleSim provide methods for model linearization. If a linearized model exists then computing and plotting the frequency response is straightforward, for example by using methods from the LinearSystems library

[1]. Nevertheless, standard linearization may yield unsatisfactory results. Imagine the case of a narrow dead zone characteristic at the system's input. Then the gain of the linearized system is zero. However, it is evident that *zero gain transfer behavior* is a deficient system description. Other examples of nonlinearities which yield problematic linearization results include hysteresis, breakaway force, slackness, and so on. In other cases system specifications directly refer to frequency responses which were determined using sinusoidal inputs with certain amplitude. An example from the domain of vehicle dynamics will be given in section 5. Again, standard linearization is not appropriate.

On the other hand recording the frequency response data in a real world or simulation experiment requires that the regarded system is stable.

1.3 Frequency response of nonlinear systems

Generalizing the frequency response from linear to nonlinear systems there is the notion of a *describing function* which will be explained in section 4. For ease of presentation, in this paper we will understand the term *frequency response* in the expanded sense of the describing function: It is the quotient of two phasors. The denominator of this quotient is the phasor of the sinusoidal input function with preassigned amplitude. The numerator is the phasor of the first harmonic of the steady state system response after all transient portions have decayed. Generally spoken, the frequency response hence is not only a function of the frequency. It may depend also on the input amplitude, cf. section 4.

1.4 Contribution of the Modelica package and this paper

Frequency response related topics having been addressed in the Modelica context so far include the investigation of powertrain oscillations in the SimulationX environment [2] and the modal approach for flexible bodies [3]. Due to the lack of a generic Modelica frequency response data acquisition tool, it occurs so far that alternative environments such as Matlab are used for post processing simulation results of Modelica models [4]. The new package presented in this article is a contribution to close the gap.

The paper is organized as follows. The issue of adequate system stimulation is addressed by section 2. The procedure used for frequency response and describing function determination is explained in 3. For convenience, in section 4 the theory of describing functions is briefly described. Section 5 is dedicated

to the Modelica specific implementation and the display of application examples.

2 Plant stimulation

For simplicity, in this paper the term *plant* stands for a Modelica model whose frequency response is searched for. Contrary to real world experiments, in this case we do not need to consider the effect of disturbances or noise. Of course, the approach is explicitly not restricted to controlled systems.

Basically, the experimental frequency response determination of a plant presumes that it is stimulated in an adequate manner. Excitation signals can be distinguished discrete valued vs. continuous, deterministic vs. random, periodic vs. step or impulse, etc. The appropriateness depends on the class of systems to be identified and the method applied for identification. In any case all system modes of interest should be stimulated to be apparent significantly enough in the output signal. For details, interested readers are referred to the technical literature, e.g. [12].

2.1 Quasi-harmonic plant excitation

Here, we confine our considerations to quasi-harmonic excitation signals, i.e. the stimulus is based on a sinusoidal function. Moreover, both the amplitude and the frequency may depend on time. Here, we follow up the concept of *sinus sweep* or *chirp* signals. The function

$$u(t) = A(t, f(t)) \cdot \sin(2\pi \cdot F(t)) \quad (1)$$

is used as plant input where

$$F(t) = \int_0^t f(\tau) d\tau \quad (2)$$

is the integral of the instantaneous frequency f . Incidentally, $F(t)$ indicates the number of elapsed periods of the sine function and is therefore called *period function* in the sequel. The amplitude A may depend on time and/or frequency.

In the simplest case, the frequency is constant:

$$f_{\text{const}}(t) = f_{\text{Start}} \quad (3)$$

$$F_{\text{const}}(t) = f_{\text{Start}} \cdot t$$

However, in this case the stimulus consists of only one single frequency. A frequency response determination covering a frequency range thus needs several separate experiments each with its own frequency value along a sufficiently fine grid. Each of the ex-

periments should take long enough such that the plant gets steady state before any serviceable response data can be collected.

More generally, with (1), (2) it is possible to stimulate the plant continuously over a range $[f_{\text{Start}}, f_{\text{End}}]$ of frequencies $f(t)$, i.e.

$$f_{\text{Start}} \leq f(t) \leq f_{\text{End}} \quad (4)$$

each with adjusted amplitude in one single experiment. However, the amplitude and frequency should be varying so slowly, that the plant can be considered in a steady state oscillation at any time. In reality this assumption cannot be assured perfectly, otherwise the total experiment would take an infinite duration. Rather, a scheduling for amplitude and frequency is searched for such that accuracy of the result is traded off against efficiency of the experiment. We start from the conception that it is most efficient to change the frequency and/or amplitude just as much as is necessary such that the transient effects remain negligible. At the same time it should be kept in mind that after plant excitation the plant response will be recorded and processed (cf. section 4) for frequency response data acquisition. Ideally for efficiency, each period contributes serviceable i.e. non-redundant data.

For illustrating the frequency scheduling problem lets assume that the sinusoidal excitation signal sweeps the interval from $f_{\text{Start}} = 0.1$ Hz to $f_{\text{End}} = 10$ Hz with a total number of $n = 10$ periods. Commonly, for frequency sweeps either linear

$$f_{\text{linear}}(t) = f_{\text{Start}} + k_l \cdot t \quad (5)$$

$$F_{\text{linear}}(t) = f_{\text{Start}} \cdot t + \frac{k_l}{2} t^2$$

or exponential

$$f_{\text{exponential}}(t) = f_{\text{Start}} \cdot k_e^t \quad (6)$$

$$F_{\text{exponential}}(t) = \frac{f_{\text{Start}}}{\ln(k_e)} \cdot k_e^t$$

frequency functions are used. Note that the parameters k_l , k_e can be chosen each such that f_{End} is reached after n periods by solving $f(t_{\text{End}}) = f_{\text{End}}$ and $F(t_{\text{End}}) = n$.

However, the plot of the period function over the logarithm of the frequency as shown in Figure 2 reveals, that in both cases, i.e. blue line for the linear case and red line for exponential the number of periods at upper frequencies is disproportionately high. In fact, too few periods are spent at low frequencies. As a result the excitation signal is warped during the

first couple of periods and cannot be considered *sinusoidal* (cf. Figure 3).

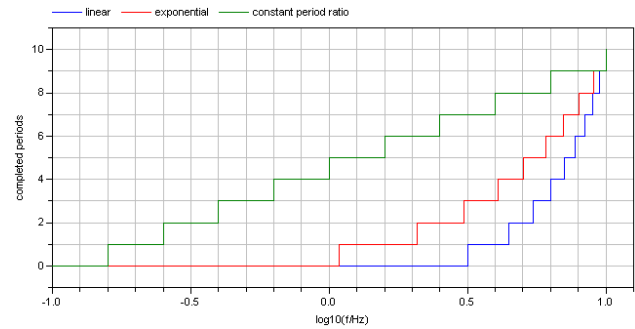


Figure 2: Plot of completed periods over logarithmic frequency.

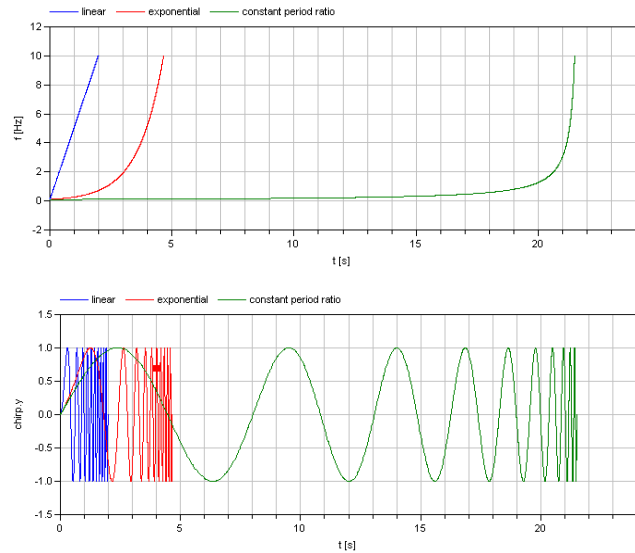


Figure 3: Comparison of sine sweep signals (lower) using different frequency functions of time (upper plot).

On the other hand, the Bode diagram as the most commonly adopted representation of the frequency response uses logarithmic frequency scaling. This suggests using a different frequency progression such that the periods are equidistantly distributed over the logarithmic frequency, corresponding to the green line in Figure 2. A frequency progression function can be derived which exhibits the just formulated property:

$$f_{\text{constant period ratio}}(t) = \frac{f_{\text{Start}}}{1 - f_{\text{Start}} \cdot \ln(k_e) \cdot t} \quad (7)$$

$$F_{\text{constant period ratio}}(t) = -\frac{\ln(1 - f_{\text{Start}} \cdot \ln(k_e) \cdot t)}{\ln(k_e)}$$

For proof let t_k be an arbitrary instant of time and t_{k+1} denote the time one period later then the correlation

$$F(t_{k+1}) = 1 + F(t_k) \quad (8)$$

holds by definition (1). Solving (8) for t_{k+1} yields

$$t_{k+1} = \frac{1}{k_c} \left(\frac{k_c - 1}{f_{\text{Start}} \cdot \ln(k_c)} + 1 \right). \quad (9)$$

Evaluation reveals that

$$\frac{f(t_{k+1})}{f(t_k)} = k_c \quad (10)$$

i.e. the frequency ratio over any period of $u(t)$ is constant independent of time, q.e.d. The frequency in fact increases exponentially with the *period function* rather than with time and each period does contribute a sample of the frequency response along an equidistant grid of the logarithmic frequency.

Figure 3 compares the corresponding time signals of the three variants of frequency progression during an experiment. In all simulations the total number of periods is $nPeriods = 10$ and the frequency sweeps from 0.1 Hz to 10 Hz, cf. Figure 2. The result reveals that with the frequency progression defined by (7) the comparatively high expense at low frequencies implicates far better sinusoidal shape but longer total simulation time.

2.2 Scheduling of excitation amplitudes

Using (1) as plant excitation signal, not only the frequency can change with time. Also the amplitude can be made varying according to one's needs as for example to reduce high output amplitudes at poorly damped system modes and thus to prevent damage to real world systems if applicable. Moreover, amplitude scheduling will be used later in the context of describing functions.

3 Frequency response calculation algorithm

As indicated in the introduction, the frequency response is understood as a *complex-valued gain* which is defined as the ratio of the complex-valued phasor of the output signal's first harmonic

$$\tilde{y}_g(\omega \cdot t, \omega) = \hat{y}_g(\omega) \cdot e^{j(\omega \cdot t + \varphi_{y_g}(\omega))} \quad (11)$$

over the complex-valued phasor of the sinusoidal input

$$\tilde{u}(\omega \cdot t) = A \cdot e^{j\omega \cdot t} \quad (12)$$

that is

$$N(\omega, A) = \frac{\tilde{y}_g}{\tilde{u}} = \frac{\hat{y}_g(\omega)}{A} e^{j\varphi_{y_g}(\omega)}. \quad (13)$$

The complex-valued gain $N(\omega, A)$ i.e. the frequency response data is tabled over sampled values of frequency and/or excitation amplitude. It contains the information about gain and phase of the plant's transfer behavior and may be displayed in a Bode diagram. Discrete Fourier transformation of input and output signals is used for its calculation. In detail, the algorithm used in the Modelica package at hand executes the following steps:

1. The plant's input is stimulated with a sinus sweep according to (1). Along the way the frequency and/or amplitude should be changing so slowly that the plant can approximately be assumed steady state all the time. Frequency progression according to one of the options (3), (5), (6), or (7) is assumed.
2. By definition periods begin when $F(t)$ takes on integer values.
3. An integer parameter $nSamples$ defines how many samples of the plant response are taken per period. Triggered by corresponding events when $(F(t) \cdot nSamples - \lambda)$ becomes integer then plant response data is sampled and collected over the current period. (The Real parameter λ can be set arbitrarily in the interval $0 \leq \lambda < 1$).

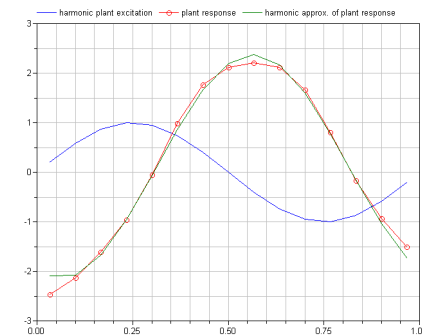


Figure 4: Exemplary signals of the elapsed period: Input signal (blue), plant response (red), harmonic approximation (green).

4. After the completion of a period the data is processed in order to identify the *complex-valued gain* assigned to the current frequency, cf. to Figure 4 for illustration. Therefore, the complex-valued Fourier coefficient of the first harmonic of the plant response is computed. It results from simple discrete Fourier transform of the time series formed by the collected data of the elapsed

period. Also the complex-valued Fourier coefficient of the first harmonic of the input signal time series is computed. (The latter depends only on A , $nSamples$, and λ and can therefore be computed beforehand without the need for sampling.) The *complex-valued gain* is the quotient of these two Fourier coefficients.

5. Procedure steps 1-4 need to be repeated for each of the grid frequencies and/or amplitudes. The *complex-valued gains* representing the frequency response data are successively stored in a table.
6. Finally, the total frequency response may be displayed in a Bode diagram and used for any of the purposes discussed in the introduction.

4 Describing functions und dual locus method

The matter of this section is adopted from [5], [6] and presented here for convenience only and to support the understanding of the application example in section 5.6.

4.1 Assumptions

Limit cycles are periodic oscillations performed by non-linear systems. Without external input signals, the oscillations sustain with a certain frequency and amplitude. Since in many cases they are not desired, criteria that enable non-linear stability analysis are useful. An approximation method that can be applied for analyzing the existence and properties of limit cycles for a class of non-linear systems is the dual locus method [9], [10]. This method requires several assumptions. The first assumption is that it is feasible to represent the open-loop system as a series connection of a single nonlinearity n and the remaining linear part $G(s)$. Therefore, the total system consists of a single loop as depicted in Figure 5.

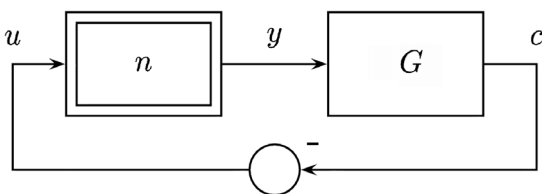


Figure 5: Closed loop formed by a nonlinear part n and a linear filter G .

The analysis of limit cycles starts with the assumption that the system is in the state of a sustained oscillation. Another assumption is necessary for the application of the dual locus method: the linear part

G needs to have distinct low-pass properties in the frequency range of the considered oscillation and at higher frequencies. As a rule of thumb, a drop of -40 dB per decade is required.

4.2 Harmonic linearization

With these assumptions it is reasonable to assume that the signal u being present at the input of the non-linearity is approximately sinusoidal. This holds because u is equal (except for the sign) to the output signal c of the linear part. Since the higher harmonics are attenuated by the low-pass effect of G , the output of the non-linearity can be approximated by its first harmonic. Thus, the consideration of the non-linearity can be restricted to its transmission of sinusoidal input signals, and a linear approximation of the non-linear system can be obtained. This approach is called *harmonic linearization*.

Furthermore, a *describing function* N of the nonlinear system n is defined according to (13) as the frequency response from a sinusoidal input signal to the first harmonic of the output signal. For static characteristics, the describing function depends only on the input amplitude A . Input and first harmonic of the output are in phase and thus $N(A)$ is real-valued. The describing function may also be applied to non-linear dynamic elements that produce a frequency- and amplitude-dependent phase shift. Then the describing function $N(\omega, A)$ is complex-valued. For some elementary non-linearities, the describing functions can be derived analytically [10] by Fourier series expansion of the periodic signal y .

4.3 Dual locus method

If the system shown in Figure 5 is in a sustained oscillation and the abovementioned assumptions hold then the transmission properties of the non-linearity can be approximated by its describing function. This leads to the condition

$$N(\omega, A) \cdot G(j\omega) = -1$$

$$\text{or } G(j\omega) = \frac{-1}{N(\omega, A)} \tag{14}$$

which is denoted *harmonic balance*. If this equation holds for a pair (ω, A) then the system is capable of performing an oscillation with this frequency and amplitude. The lower representation of (14) provides the foundation for the graphical *dual locus method*: limit cycles are possible if there exist intersection points between the locus $G(j\omega)$ of the linear part and the locus of the *negative-inverse describing function* $-1/N(\omega, A)$. From the parameterization of both loci at

the intersection, the values of ω and A can be determined as properties of the corresponding limit cycle. For a criterion reflecting the stability of limit cycles the reader is referred to [6].

4.4 Example: Describing function of a rate limiter

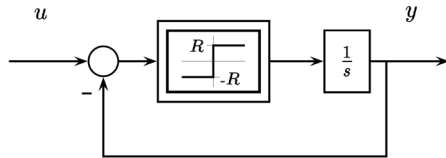


Figure 6: Ideal rate limiter.

As an example Figure 6 shows an ideal realization of a rate limiter. (Note that this realization is not suitable for numeric simulation since the infinite gain of the switch induces chattering. A remedy is replacement of the switch by a limiter with high gain.) In Figure 7 some time responses of the rate limiter are shown. Various sinusoidal input signals are applied with different frequency ω and amplitude A .

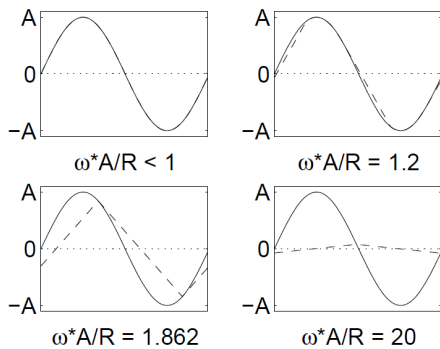


Figure 7: Input and output of a rate limiter at various values of $\omega A/R$.

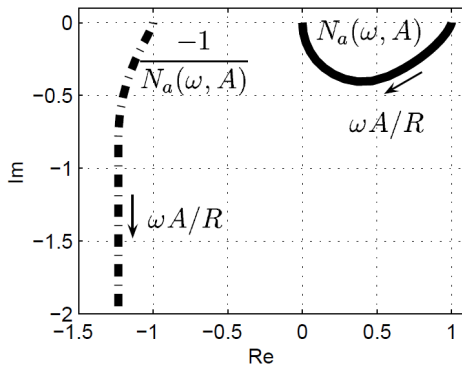


Figure 8: Locus (solid line) and negative inverse locus (dashed line) of the rate limiter describing function.

Due to the memory of the integrator this dynamic non-linearity is not representable by a static characteristic. Nevertheless, a describing function can be derived. The shape of the output signal in relation to the input signal only depends on the ratio $\omega A/R$. Therefore, the describing function only depends on this composed parameter.

Figure 8 shows both the locus and negative-inverse locus of $N(\omega, A)$. More details of the rate limiter describing function derivation can be found in [11], [6].

5 Application of the Modelica package and examples

In this section, the Modelica package application is shown in the context of Dymola 7.4 used as modeling and simulation environment.

5.1 Stimulus signal generator implementation in Modelica

A model class *Chirp* which can be used to create the plant stimuli described in section 2.1 is included in the Modelica package. By parameter settings it allows for choosing adequate frequency progression as well as signal amplitude scheduling according to one's needs.

5.2 Implementation of the algorithms for frequency response data recording

In Figure 1 the standard application is shown. The model class *FR_Recording* covers the total functionality necessary for recording the frequency response data of the block *plant*. To avoid confusion note that the interconnection shown here is *not* a closed loop in the classical meaning, since the output of *FR_Recording* does not depend on its input. Rather, stimulation and analysis functionalities are combined in one model class for easier use. The output of *FR_Recording* is internally connected to the output of a *Chirp* instance thus providing the stimulus for the plant. To complete the interconnection the plant response is input to the *FR_Recording* instance in order to make it accessible there for analysis. That is then performed according to the steps demonstrated in section 3.

Before starting the simulation, the Dymola *Experiment Setup* needs to be well defined. Only variable step solvers should be used since the events triggered by the procedure e.g. for data sampling are not equidistant in time. This is due to the continuously changing frequency but equidistant sampling along the *period function*.

Attention should be paid to the *Output→Store* settings. Depending on the model complexity storing all variables may easily exceed the available memory or disc space. Therefore, it is recommended to only store *Output* variables only at events (no *Equidistant time grid* storage). An instance of the *FR_Bus* model class should be used and connected as shown in Figure 1. On the bus all signals are available which are relevant for the frequency response evaluation. They all have the *output* prefix and will therefore be exclusively saved during simulation and be available in the Dymola variable browser, cf. Figure 9. In addition, for large models consider to set the *protected* attribute to both the plant and the *FR_Recording* instance to prevent excessive data storage.

Variables	Unit	Description
FR_PT2 1		
fr_Bus		
<input type="checkbox"/> plantExcitation	m	Plant stimulus
<input type="checkbox"/> excitationAmplitude		Amplitude of plant excitation
<input type="checkbox"/> f	Hz	Frequency [Hz]
<input type="checkbox"/> log10f		log10 of frequency/Hz
<input type="checkbox"/> plantResponse[1]		Plant response
<input type="checkbox"/> normalizedPlantResponse[1]		Normalized plant response (=plantResponse./gain)
<input type="checkbox"/> Re[1]		Real part of frequency response
<input type="checkbox"/> Im[1]		Imag part of frequency response
<input type="checkbox"/> gain[1]		Frequency response: Gain
<input type="checkbox"/> gaindB[1]	dB	Frequency response: Gain [dB]
<input type="checkbox"/> phase[1]	rad	Frequency response: Phase
<input type="checkbox"/> offset[1]		Determined offset of plant response signal
<input type="checkbox"/> tEnd	s	Maximum simulation time
<input type="checkbox"/> error[1]		Summed error squares w.r.t. sinusoidal reference signals
<input type="checkbox"/> periods		Periods of the sine function used for plant excitation

Figure 9: Dymola variable browser showing signals on fr_Bus

Figure 10 shows the dialog window for setting the parameters of the frequency response data recording tool. With parameter *m* the signal dimension of the plant output is set, e.g. *m* = 1 for single output plants. For the chirp stimulus *fStart* and *fEnd* define the frequency interval, *nPeriods* is the total number of sinusoid periods. One of four frequency progression types corresponding to (3), (5), (6), or (7) can be selected. The dependency of the stimulus amplitude on time and/or frequency is provided by a replaceable model, which allows the user to define it according to his needs. *nSamples* is used to specify the number of data points per period to collect time series data for discrete Fourier transform. Finally, some parameters can be specified to control the saving of frequency response data to disc. After the simulation signals for plotting one of multiple representations of the frequency response are available in the Dymola variable browser as shown in Figure 9.

5.3 Example: Frequency response of a mass-spring-damper system

The mass-spring-damper plant depicted in Figure 11 is used as an example to demonstrate how frequency response data can be recorded by means of a simulation while using classes from the Modelica package.

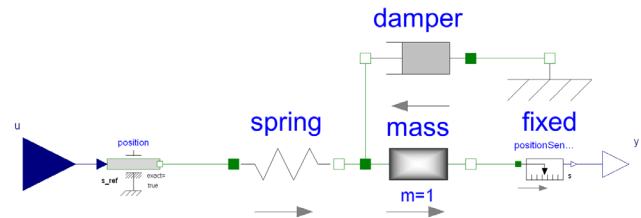


Figure 11: Mass-spring-damper system.

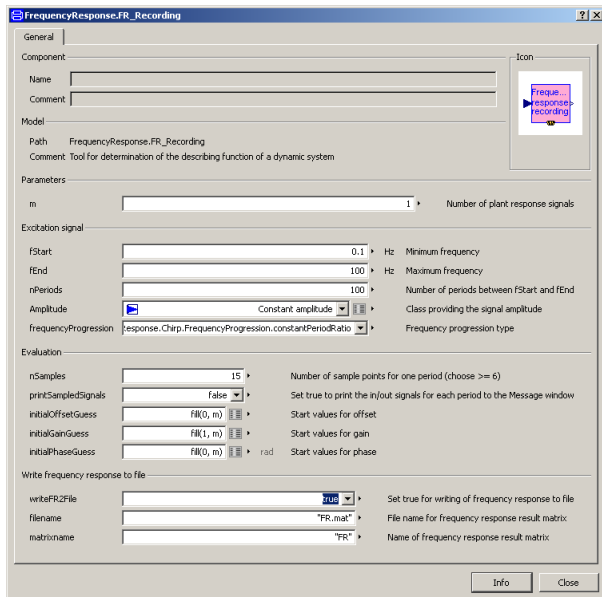


Figure 10: Dymola dialog for setting parameters of the *FR_Recording* class.

Therefore, the block *plant* in Figure 1 is made an instance of the mass-spring-damper model. It can just as well be modeled as a second order transfer function

$$G_{msd}(s) = \frac{y(s)}{u(s)} = \frac{\omega_0^2}{s^2 + 2 \cdot D \cdot \omega_0 \cdot s + \omega_0^2} \quad (15)$$

Thus, the frequency response can be calculated analytically by replacing $s = j\omega$ and the experimental result can be compared to this precise analytic reference (denoted *theory* in the plots below). The parameters of the mass-spring-damper plant are chosen such that the resonance frequency is $\omega_0/2\pi = 10$ Hz and the damping coefficient is $D = 0.2$.

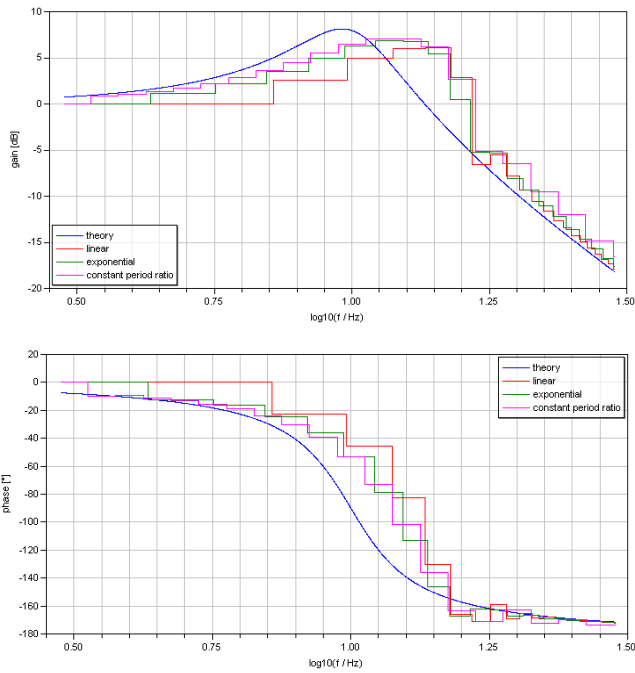


Figure 12: Influence of frequency progression on the resulting Bode plot

In Figure 12 first attempts are shown to determine the frequency response of the mass-spring-damper system using only twenty excitation periods. The simulation is repeated with each of the three frequency progression variants (5), (6), (7). With the *constant period ratio* frequency progression according to (7) the Bode plot frequency grid points are equally distributed along the logarithmic frequency axis which appears to be the better choice when compared to linear (5) or exponential (6) progression. Figure 13 continues the simulations shown in Figure 12. However, now the total number of periods n is increased from one simulation to the next whereas the frequency progression with *constant period ratio* is kept. The comparison shows that the accuracy of the resulting frequency response data clearly improves with increasing $nPeriods$ and converges towards the analytic reference. The reason is that the transient portion of the plant response loses significance when the frequency is changing at a lower rate. In practice the precise reference normally is not known. Then, a reasonable number of periods can be found by gradual increase until the change of the result appears tolerable. In addition, at least for linear systems the plant response offset and/or the deviation between the plant response and its first harmonic can be observed to find out whether the system is sufficiently steady state. Both quantities are also calculated by the tool after each completed period.

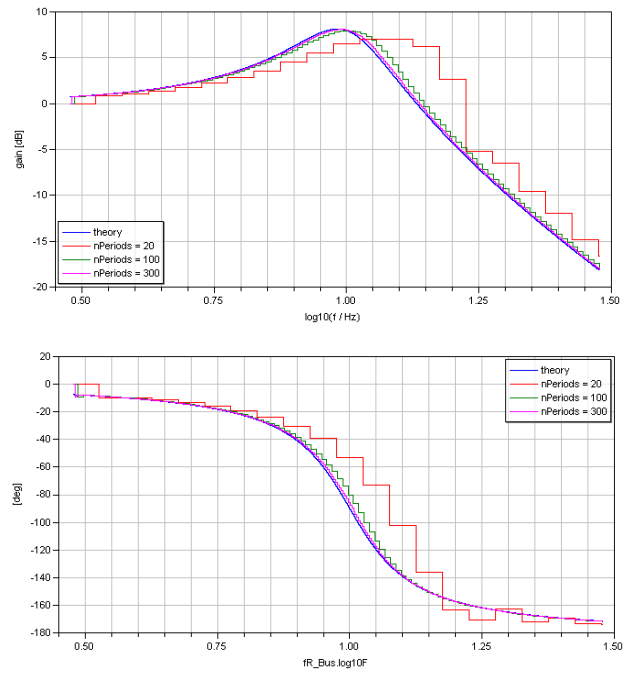


Figure 13: Influence of the number of periods $nPeriods$ on the resulting Bode plot.

5.4 Example: Frequency response of a complex multi-body vehicle model

The recording of frequency response data using the Modelica package does not only work for simple academic models.

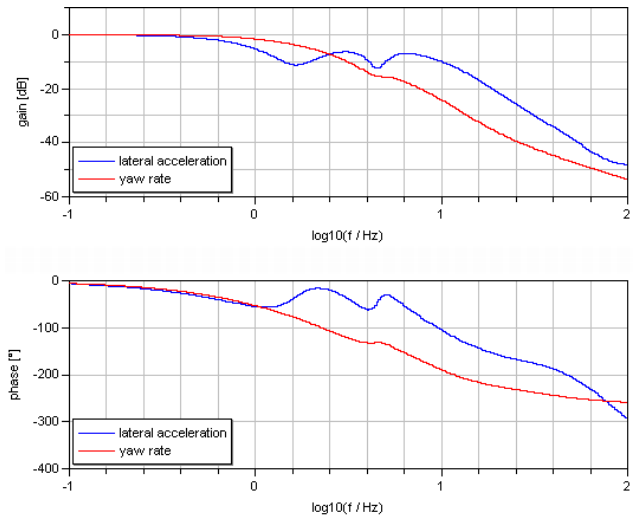


Figure 14: Bode plot of a complex multi-body vehicle model at 80 km/h; The input is the steering angle, outputs are lateral acceleration (blue curve) and yaw rate (red curve).

In Figure 14 the analysis of a multi-body vehicle model with 54 states from our *VehicleControls* library [7] is shown. The comparison of the vehicle

steering response w.r.t. gain and phase of yaw rate vs. lateral acceleration allows for assessment of respective criteria [13]. The gains are normalized with the respective steady state gains here.

The frequency response data is represented by nice smooth curves even in face of the complexity and nonlinearity of the plant. *Dassl* was used as integration method. The total simulation of 434 seconds took 169 seconds on a 3.0 GHz MS Windows PC.

5.5 Frequency response of multi-input multi-output (MIMO) models

Multiple plant outputs can be handled without any modification but setting the number m of plant outputs correctly, cf. Figure 10. The frequency response data for each frequency sample will then be vector valued, correspondingly.

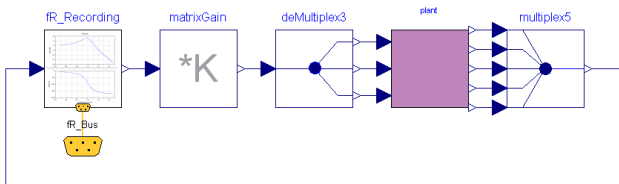


Figure 15: Total Modelica model for frequency response recording of a MIMO plant model.

Handling of multiple inputs is no difficulty either. Apparently a stimulation of multiple inputs at the same time is not expedient. Hence the analysis should be performed by stimulating one input only in one simulation. Figure 15 shows the frequency response data recording setup for an exemplary MIMO plant model exhibiting five outputs and three inputs. The active input can be selected by appropriate choice of the *matrixGain.K* parameter vector.

5.6 Application example for describing functions

The tool at hand can be applied to determine describing functions in exactly the same manner as for frequency response data recording. The example shown here examines the rate limiter from section 4.4. Therefore, the *plant* instance in Figure 1 is redeclared by the corresponding rate limiter Modelica class (using $R = 1$ here). The combined parameter $\omega A/R$ is the only relevant independent quantity, see section 4.4. Therefore we can choose the variation of either ω or A . In this example we choose ω constant by setting $fStart = fEnd = 1\text{Hz}$ and specify a variation of A with time, instead. (This proceeding is suitable especially for all static non-linearities.) The re-

sults are shown in Figure 16 in terms of the Bode diagram. The describing function may as well be represented as a locus or negative inverse locus, which was already shown in Figure 8.

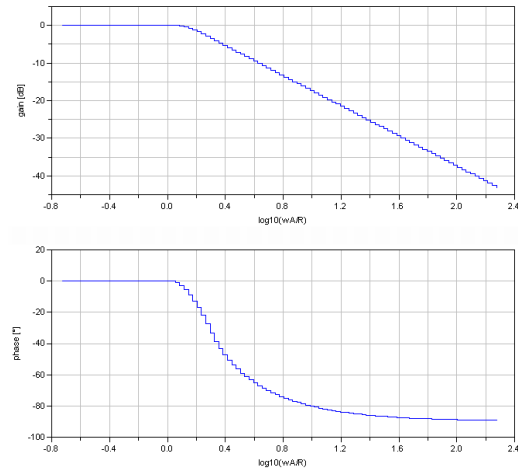


Figure 16: Bode plot of rate limiter describing function.

Now, the dual locus method is applied to investigate whether the closed loop shown in Figure 17 can perform limit cycles. Here, the linear system model is the second order lag system already considered in section 5.3, however, with a damping coefficient of $D = 0.1$ here.

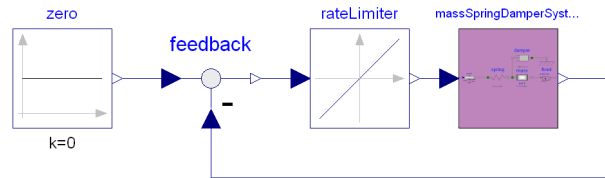


Figure 17: Closed loop with second order linear system and rate limiter in series connection.

Figure 18 shows both the negative inverse describing function locus of the rate limiter (green line) and the locus of the linear part. Again the analytical result (blue line) and the recorded frequency response data (red line) are compared. The discrepancy between them is noticeable. This is due to the specific graphical representation. With the locus, the highest gain (which occurs at the resonance peak, compare Figure 13) appears most prominent. Obviously, during frequency response data recording the system would need some more time here to become steady state and to produce a preciser locus.

Two intersection points between the Nyquist locus of the linear part and the locus of the negative inverse rate limiter describing function exist. Further analy-

sis yields that only the lower intersection point represents a *stable* limit cycle [6].

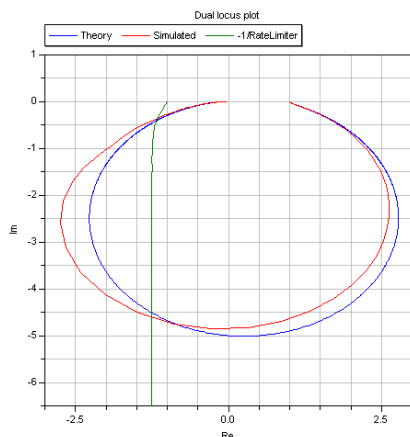


Figure 18: Dual locus plot for the detection of limit cycles of the system shown in Figure 17.

A time simulation of the total system from Figure 17 with dedicated initial condition in fact exhibits the limit cycle shown in Figure 19. The frequency and amplitude of the limit cycle correspond to the parameterization of the loci at the lower intersection point.

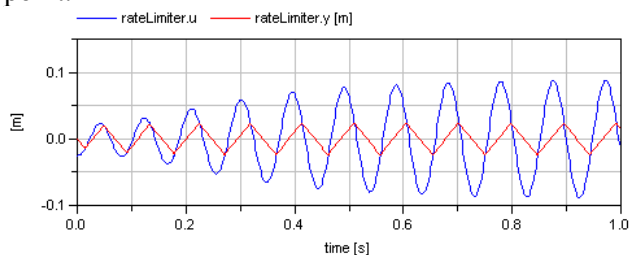


Figure 19: Limit cycle performed by the system shown in Figure 17.

5.7 System identification from frequency response data

To complete the application examples, briefly the option of identifying LTI models (linear time invariant models) from frequency response data is discussed. Imagine the task to derive a handy LTI representation of a rather complex nonlinear Modelica model when application of the linearization method (cf. section 1.1) is not reasonable. This may be the case e.g. due to one of the problems illustrated in section 1.2. The LTI representation can be used e.g. for model based control. Moreover, models represented in terms of frequency response data cannot be simulated and therefore, back-translation into a model which can be numerically time-integrated may be useful. For example, it may serve as an alternative

complexity reduced model in a simulation framework where scalable models are beneficial.

As stated in the introduction, the frequency response data generated with our package well complies with *FRD models* in Matlab. Hence, the methods available with the Matlab System Identification Toolbox (e.g. the *pem* function) can be used to derive LTI models of user-preassigned order. Another way which completely avoids Modelica foreign tools is the following: Choose an LTI model representation from the LinearSystems library [1] and an adequate system order. In the next step optimize (i.e. identify) the parameters of the LTI model. This can be done e.g. by using the model calibration feature from the Dymola Design Library such that the Bode diagram fits the frequency response data sufficiently well.

6 Conclusions

An easy-to-use tiny Modelica package for automatic recording of frequency response data and describing functions was presented. The frequency response representations which are resulting from quasi-single frequency harmonic stimulation are significantly smoother than what can be obtained from spectral analysis after stimulation with a multi-frequency (e.g. noise) input signal.

The question into which greater library this package will be usefully integrated and what needs to be done for seamless assembly still needs to be resolved.

7 Acknowledgement

The presented results were compiled in the context of the project FAIR (Fahrwerk-Antrieb-Integration ins Rad) [8]. Frequency responses of multiple vehicle concept models were to be generated for comparative assessment in one of the work packages. On behalf of the FAIR project team the author wishes to express our gratitude to *Bayerische Forschungstiftung* for funding.

References

- [1] Baur, M., Otter, M., Thiele, B.: Modelica Libraries for Linear Control Systems. Proc. 7th Modelica Conference, Como, Italy, 2009.
- [2] Abel, A., Nähring, T.: Frequency-Domain Analysis Methods for Modelica Models. Proc. 6th Int. Modelica Conference, Bielefeld, Germany, 2008

- [3] Heckmann, A. et. al: The DLR FlexibleBodies library to model large motions of beams and of flexible bodies exported from finite element programs. Proc. 5th Int. Modelica Conference, Vienna, Austria, 2006.
- [4] Edrén, J. et. al.: Modelica and Dymola for education in vehicle dynamics at KTH. Proc. 7th Int. Modelica Conference, Como, Italy, 2009.
- [5] Ackermann, J., Bünte, T.: Robust prevention of limit cycles for robustly decoupled car steering dynamics. *Kybernetika*, 35(1):105-116, 1999.
- [6] Ackermann, J. et. al.: Robust Control: The Parameter Space Approach. Springer, London, 2002.
- [7] EUROSYSLIB Project Profile, http://www.itea2.org/public/project_leaflets/EUROSYSLIB_profile_oct-07.pdf, 2007.
- [8] FAIR Project profile <http://www.forschungsfstiftung.de/index.php/Projekte/Details/FAIR-Fahrwerk-Antrieb-Integration-ins-Rad.html>, 2009.
- [9] Gelb, A., Vander Velde W.: Multiple-Input Describing Functions and Nonlinear System Design. New York: MacGraw-Hill, 1968.
- [10] Siljak, D.: Nonlinear systems: the parameter analysis and design. New York: Wiley, 1969
- [11] Duda, H.: Fliegbarkeitskriterien bei begrenzter Stellgeschwindigkeit. Ph.D. thesis, Technische Universität Braunschweig, April 1997. Forschungsbericht 97-15, Deutsche Forschungsanstalt für Luft- und Raumfahrt e.V., Köln.
- [12] Pintelon, R., Schoukens, J.: System identification: a frequency domain approach. IEEE Press, New York, 2001.
- [13] Mitschke, M., Wallentowitz, H.: Dynamik der Kraftfahrzeuge. 4. Auflage, Springer-Verlag, 2004.

On using model approximation techniques for better understanding of models implemented in Modelica

Anton Sodja Borut Zupančič

Faculty of Electrical Engineering, University of Ljubljana
Tržaška 25, 1000 Ljubljana, Slovenia
{anton.sodja, borut.zupancic}@fe.uni-lj.si

Abstract

Modelica enables rapid development of detailed models of heterogeneous and complex systems. However, resulting models are as complicated as reality itself and therefore it may be hard to identify causes for model behavior or verify that model behaves correctly. A traditional engineering approach is to use intuition and experience to identify important parts of the model with the highest impact on model behavior for specific scenario. Numerous model order reduction and simplification techniques (i.e., metrics used by these methods) have been developed to automatically estimate important parts of the models for a certain scenario and thus alleviate reliance on subjective factors, i.e., intuition and past experience.

In this paper are discussed model order reduction and simplification techniques (e.g., metrics used by these techniques for rankings of elements) which are applicable to wide range of Modelica models built from already available libraries. Modelica models are translated to set of differential-algebraic equations and for the latter there are numerous tools for model order reduction already available. However, these tools are not designed for helping users understand the model's behavior and the reduced model may be hard to understand by the user because the structure of the original model is lost. Hierarchical decomposition of the model must be preserved and if the model is developed with a graphical schematics then elements (nodes) of the schematics must be ranked. Therefore we adapted energy-based metrics used in ranking of bond-graphs' elements to much more loosely defined Modelica's schematics, so they can be used complementary with ranking methods that work with equations.

Keywords: model order reduction; model simplification; verification

1 Introduction

An important aspect of the Modelica language design is user interaction for efficient modeling of large, complex and heterogeneous physical systems.

Models are usually decomposed in several hierarchical levels. On the bottom of hierarchy are submodels of basic physical phenomena which are most commonly stated as a set of (acausal) differential-algebraic equations and it is thus most conveniently that these equations can be entered directly (e.g., without a need for any kind of manipulation or even transformation to some other description formalism). On higher hierarchical levels, model is described graphically by schematics (i.e., object diagrams) and the obtained scheme usually reflects the topology of the system. Model representation in Modelica is thus understandable also to domain specialists unfamiliar with computer simulation of dynamic system.

Modelica is object-oriented modeling language and thus includes features such as inheritance and replaceable models. This language features are necessary for efficient implementation of model libraries [13], but they increase implementation complexity and make browsing sources of the components from libraries more difficult. For example, component *DynamicPipe* – a model of a straight pipe with distributed mass, energy and momentum balances – from the Standard Modelica Library consists of four base models and three replaceable elements which are also models with complex inheritance hierarchy. The description of the pipe's dynamics, equations of balances and thermodynamic state of the medium in the pipe, is split among more than ten (partial) models to achieve efficient component reuse and prevent code duplication. This kind of model decomposition might not have a physical meaning – it only addresses implementation issues.

In practice, domain specialists usually already have

some calculations (e.g. in Excel) which they want to use for verification of the model implemented in Modelica and also for clarifying unexpected behavior of (usually) much more detailed and complex model in Modelica. So, a matching between calculations they have and model in Modelica is desired. However, modeling environments supporting Modelica currently do not provide many tools that would facilitate investigating and exploring the model. Most of the complex models are build up with use of different model libraries and when the documentation of those libraries do not suffice, especially when the submodels are highly customized components (with replaceable sub-components and modifications), it is necessary to look under the hood of the used components. But due to complicated implementation of library components, it is undoable for most domain specialists.

Engineers use experience and intuition to determine important parts of the model which have the highest impact on system's dominant dynamics or model's simulation response in specific scenario. In an attempt to diminish reliance on subjective factors such as experience, numerous modeling metrics and methodologies have been developed. They usually require strict modeling formalisms and thus not much effort was put into integrating them into Modelica environments.

2 Model order reduction and simplification techniques

Detailed models of complex systems are also as complex and hard to understand as reality they model. The interpretation of underlying equations or extraction of an in-depth system understanding can get impossible even for relatively small systems [11]. Therefore, symbolic analysis methods, most notably of electrical circuits, incorporate various symbolic approximation techniques which are used to simplify symbolic expression or schematic diagrams and also reduce order (state-space dimension) of the model [10].

An important class of the model order reduction and simplification methods when used in system analysis or for structural design is when they generate a *proper model*, i.e., reduced model with the minimum complexity required to meet the performance specifications and possessing physically meaningful parameters and states [5].

A numerous mixed numerical-symbolic model order reduction and simplification techniques have been developed and successfully applied so far [10, 12, 4,

5]. They usually consist of running a series of simulations, ranking the individual coordinates or elements by the appropriate (quantitative) metrics and removing those that fall below a certain threshold [2].

2.1 Equation-based simplification

All analytic models can be described by a system of equations and even if some other modeling formalism is used (e.g., block schemes, bond graphs, etc.), it is possible to export the model as a system of equations. However, model representation in a form consisting of symbolic (algebraic) expressions is meaningful to user only in certain situations, for example, use of transfer functions in control design.

For equation-based simplification, variables of interest must be selected and metrics used for ranking of expressions' terms is then selected as a numerical error with respect to an objective function given by the variables of interest.

Simplification strategies include various algebraic manipulations (e.g., substitution of a variable), where no error is introduced into the simplified equations, and modification of the equations that results in the approximate system (e.g., term deletion, linearization of equations, etc.) which requires a numeric simulation to determine the error caused by modification [14].

Simplification can have a global effect, i.e., affects whole system of equations, when some variables of the system are manipulated or local effect when only single term of one equation is manipulated.

2.2 Structure-based simplification

Most of modern modeling tools provide a graphical interface where models are represented by schematics. Graphical descriptions of the models are based on various modeling formalisms, schematics can be a merely graphical representation of algebraic expressions (e.g., block graphs) and symbolics comprising the schematics represent single or a group of algebraic operations or they can provide additional information about the system (e.g., information about topology of the system). In the latter case, it is sensible to chose customized simplification techniques, although it is possible to map models simplified by equation-based order reduction and simplification techniques to a graphical representation of the original model [12].

Because all physical systems have in common conservation of mass and energy, a widely used class of metrics for order reduction of proper models in physical-systems modeling are related to energy or

power [2]. Energy-based metrics require a modeling formalism where energy of the model's components is easily extracted, for example, bond graphs [9, 5]. Bond-graph modeling is a form of object-oriented physical systems modeling: elements can be seen as object interacting with each others – interactions are described by acausal bonds [1].

Among successfully applied energy-based techniques for bond-graph simplification are ranking of elements based on RMS power of bonds [9], ranking on activity – amount of energy that flow in and out of the element over the given time [5] and comparing the energy associated with each bond to those in neighboring bonds and eliminating those with smallest relative energy [15]. Result of a model simplification by these techniques is also a model described by bond graph. Furthermore, all the energy-based metrics have some physical meaning and can thus help with understanding and addressing modeling issues.

3 Simplification of models in Modelica

According to authors knowledge, there is no modeling environment that provides tools for simplification and order reduction of model implemented in Modelica directly. A Modelica model must be flattened and the resulting DAE system is then exported to a designated tools where model order reduction and simplification is performed.

This approach is suitable for some applications, for example, when reduced model is needed for control design. In such cases, loss of information caused by flattening is not problematic, because only close matching of reduced and original model's behavior is required. However, in applications like model verification and debugging or when model is used to gain insight for system performance improvement, it is desired that simplified model is also a valid Modelica model with the same structure as the original (with the same hierarchical decomposition and topology of schematics).

4 Ranking elements of object diagram

4.1 Choice of metrics

Object diagrams consist of connected symbols representing components (submodels). What kind of in-

teraction a connection defines is determined by type of connectors (i.e., ports) the connected components have. In Modelica is a type of connector very loosely defined. In general, it is a list of variables with some qualifications (e.g., causality, type of variable: intensive – extensive, etc.), but it can also have a hierarchical structure [6].

Although a large number of different kind of schematics can be modeled with appropriately defined connectors, are the most important acausal connections for modeling physical interactions. Each (dynamic) interaction between physical systems results in a energy exchange between the system, so it is very intuitive to chose energy-based metrics for simplification of physical systems models.

Modelica's object diagrams, when modeling physical systems, share some similarities with bond graphs, which are also a form of object-oriented acausal modeling. Therefore it is easy to adapt most of bond-graph simplification techniques to Modelica's object diagrams.

Connectors usually contains a pair of effort and flow variable (however, their product is not necessarily an energy flow like in bond graph formalisms), as can be seen by inspecting Modelica Standard Library [7] where elementary connector definitions for almost all physical domains are gathered:

- Interaction between components in analog circuits (*Modelica.electric*) is determined by voltage v and current i , the latter is a flow variable, and the power of the interaction is product of both variables: $p = v \cdot i$.
- Similar is connector in *Modelica.Magnetic* composed of variables for magnetic potential difference V_m and magnetic flux Φ , an effort and flow variable respectively. Power of the connection is product of variables: $p = V_m \cdot \Phi$.
- Connectors used for modeling of 1-D mechanics, translational and rotational, consist of position s and angle ϕ respectively, and force f and torque τ respectively. However, product of connector's effort and flow variable is no longer power. For determination of the power of connection, displacement variable has to be differentiated: $p = \frac{d}{dt}s \cdot f$ and $p = \frac{d}{dt}\phi \cdot \tau$ for translational and rotational mechanics respectively.
- In *Modelica.Multibody* library, which deals with 3-D mechanics, are effort and flow variables no longer scalars, they are 6-dimensional vectors,

so a state of a free-body (having 6 degree-of-freedom) can be determined. Furthermore, due to computational restrictions, implementation of connector takes also into account a suitable selection of a frame of reference (forces, torques and orientation are expressed in local, while position is in global frame of reference). A definition of the connector is the following:

```
connector Frame
  SI.Position r_0[3];
  Frames.Orientation R;
  flow SI.Force f[3];
  flow SI.Torque t[3];
end Frame;
```

Position is determined with variable r_0 , while orientation R is a structure containing transformation matrix T from global to local frame of reference and vector of angular velocities ω in local frame of reference. Forces and torques are given by vectors f and t respectively. Power of the connection can be calculated by expression: $p = \frac{d}{dt}(\mathbf{T} \cdot r_0) \cdot f + \omega \cdot t$, where again, there is a need to differentiate position after transformation to local frame.

- Connector for modeling heat transfer in 1-D consists of effort variable temperature T and flow variable for heat-flow rate Q_{flow} . The energy transfer is in this case equal to flow variable, $p = Q_{flow}$.
- Library *Modelica.Fluid* deals with modeling of heat and mass transfer. The connector used in library's components which covers also mass transfer is implemented as following:

```
connector FluidPort
  replaceable package Medium =
    Modelica.Media.Interfaces.PartialMedium;

  flow Medium.MassFlowRate m_flow;
  Medium.AbsolutePressure p;
  stream Medium.SpecificEnthalpy
    h_outflow;
  stream Medium.MassFraction
    Xi_outflow[Medium.nXi];
end FluidPort;
```

Besides effort and flow variable, pressure p and mass-flow rate m_{flow} respectively, the connector includes also additional information about properties of the substance which is being exchanged in the interaction modeled by a connection of type *FluidPort*: specific enthalpy h and composition of substance (vector of mass fractions X_i if substance is a mixture). The thermodynamic state

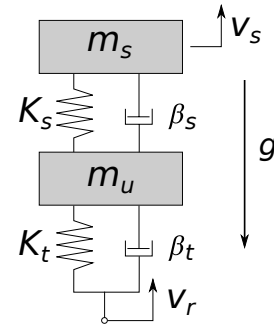


Figure 1: Scheme of car a suspension.

of the substance is uniquely determined by the variables of connector and all the other (thermodynamic) properties can be calculated by using functions provided by package *Medium* which is a parameter of the connector. However, thermal diffusion is not covered by this connector (it is neglected).

Energy flow associated with the connector is composed of thermal, hydraulic and chemical term and could be calculated as following [3]: $p = \dot{m} \cdot s \cdot T + \dot{m} \cdot p / \rho + \sum \mu_i \cdot \dot{N}_i$. Quantities specific entropy s , temperature T , density ρ , chemical potential μ_i and molar flow \dot{N}_i can be calculated from thermodynamical state equations provided by package *Medium*.

Although it is possible to calculate energy flow of the connector from the variables of the connector, this is not always possible to do as a post-processing the simulation results. For example, derivative of the position or angle in connector of the library for 1-D mechanics may not be available if this variable is not chosen for state variable. This implies instrumentation of the model.

Most bond-graphs energy-based metrics, like [5], require energy flow of the element. Fig. 1 illustrates a scheme of a car suspension for one wheel. Corresponding representation of a model with a bond graph is depicted in Fig. 2. In Fig. 2 can be seen that each element (e.g., tire stiffness) is represented with a bond which have an element symbol on one end and with another it is connected to the 1-junction. A model of the car suspension from Fig. 1 build from Modelica Standard Library's components is shown in Fig. 3. Because bond-graph and Modelica's object diagram preserve system topology, there are some analogies between them. A connection node in Modelica, when two or more connectors are connected together), is equivalent to 0-junction in bond-graph representation

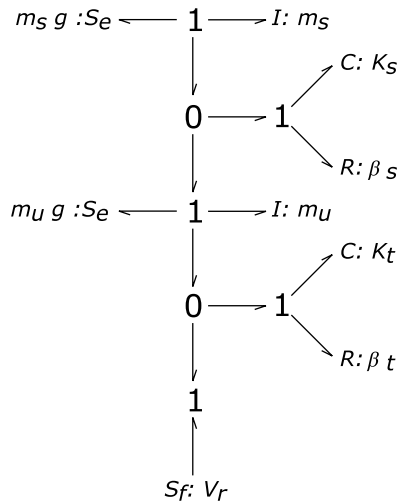


Figure 2: Bond graph of a car suspension

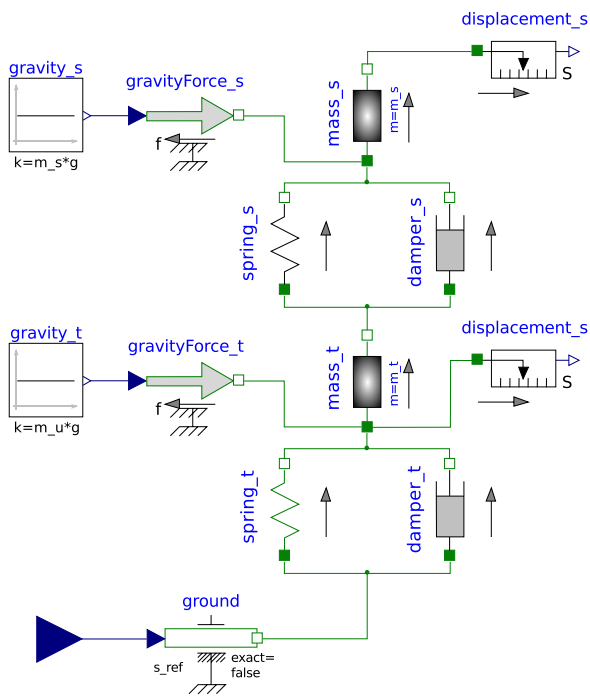


Figure 3: Car-suspension model represented by a Modelica object diagram.

– the effort variables of connected connectors are the same and the flow variables sum to zero. According to the analogy, each Modelica’s component in object diagram should be analogous to 1-junction – effort variables of the component’s connectors should sum to zero. However, this is not the case in Fig. 3 where effort variable defined in connectors is absolute position. Nevertheless, energy conservation law implies that energy flow of all component’s connectors and change of energy stored, added or removed by component must sum to zero. Therefore, the energy flow which corresponds to energy flow of bond represent-

ing the element can be in Modelica object diagrams calculated as sum of energy flows of the component’s connectors. For example, activity A of a component, weighting factor used in metrics proposed by [5], is thus calculated for Modelica components as following:

$$A = \int_{t_1}^{t_2} \left| \sum_{i=0}^{N-1} -p_i(t) \right| \cdot dt \quad (1)$$

In Eq. 1, $p_i(t)$ designates power flow into i -th connector, N is the number of connectors in component and $[t_1, t_2]$ is the time window of observation.

4.2 Model instrumentation

In order to assure that all the necessary data for selected (energy-based) metrics evaluation are provided in the simulation results, model must be instrumented, i.e., additional equations must be inserted into the model.

It is possible to insert equations for weighting factors (e.g., Eq. 1) directly. However, this introduces many new equations and states into the simulation model and can have a very negative impact on simulation’s duration and also on numerical stability. This can be problematic especially with large models, where the use of model approximation methods is the most sensible.

Therefore we decided to use the least instrumentation possible and do most of calculations of weighting factors as a post-processing of simulation results. Model instrumentation was implemented in Open-Modelica’s shell [8]. Before model can be simulated, it must be loaded into the environment together with all the libraries it requires. Upon loading, abstract syntax tree (AST) of the model is generated and saved into the environment. So, instrumentation was implemented as a separate function which traverses the AST in the environment and for each connection encountered inserts an equation for energy-flow calculation. What kind of equation needs to be inserted is determined by inspecting the type of connector used in the connection equation. For this purpose, a special library of components is provided to the instrumentation function. Each component of the library have as an annotation provided a fully-qualified path to the connector-type definition of which equation for an energy-flow calculation provides. If there is no corresponding component found in the library for the connection’s connector-type, that connection is skipped.

Besides instrumentation of the model, connection graphs for each hierarchical level of the models are

added to the environment.

After a model is instrumented, it can be simulated in the usual way (with command *simulate()*).

4.3 Ranking and presentation of results

Ranking of components is performed as post-processing of simulation results. This enables possibility of switching ranking metrics without repeating (possibly time-consuming) instrumentation and simulation. Furthermore, ranking of components which are not of current interest can be avoided.

In our current implementation, activity-metrics (Eq. 1) is used for ranking. Each hierarchical level is considered separately. To determine activity of the component, a connection graph of the object-diagram on given hierarchical level is taken from environment (where it was put by instrumentation function) and energy flows of component's connections are extracted from it. The absolute sum of connection's energy flows (as determined by Eq. 1) are numerically integrated by a quadrature form. However, because integration is done as post-processing, there is a significant loss of accuracy. Quadrature formulas have a much higher truncation-error than solvers used for simulation of the model. Furthermore, such integration is affected by the chosen communication interval. Nevertheless, because accuracy of weighting factors is not of critical importance, use of quadrature formulas suffice in most cases.

The results of ranking are currently provided only in printed form (in a tableau), because there was no graphical interface suitable for adaptation available. Simplification of the model based on obtained ranking is not implemented yet.

Element	activity [J]	relative [%]	accumulated [%]
gravityForce_s	2,270.06	37.06	37.06
spring_s	1,763.33	28.79	65.85
ground	795.02	12.98	78.82
mass_s	787.65	12.86	91.68
damper_s	198.82	3.25	94.93
spring_t	192.57	3.14	98.07
gravityForce_t	92.98	1.52	99.59
mass_t	24.53	0.40	99.99
damper_t	0.53	0.01	100.00
displacement_s	0.00	0.00	100.00
displacement_t	0.00	0.00	100.00

Table 1: Ranking of components when model from Fig. 3 is given input shown in Fig. 4.

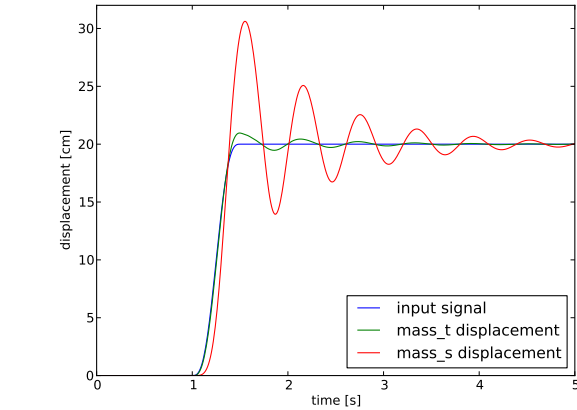


Figure 4: A car hits a smooth curb: low-frequency excitation signal is given as an input to model on Fig.3. Also a response – displacement of a unsprung (*mass_t*) and sprung mass (*mass_s*) is depicted.

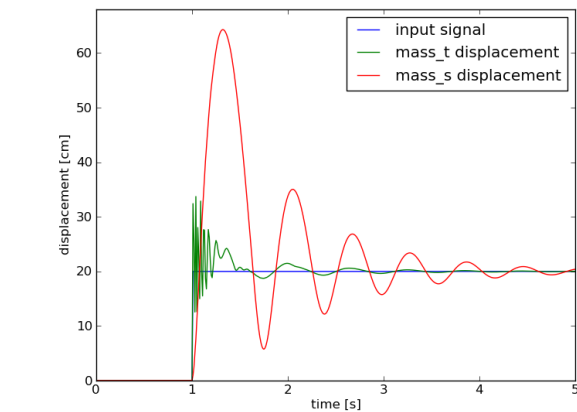


Figure 5: A car hits a sharp curb: step signal is given as an input to model on Fig.3. Also a response – displacement of a unsprung (*mass_t*) and sprung mass (*mass_s*) is depicted.

Element	activity [J]	relative [%]	accumulated [%]
mass_t	528,914.61	43.25	43.25
spring_t	481,340.66	39.36	82.61
damper_s	115,233.47	9.42	92.03
spring_s	49,128.08	4.02	96.05
damper_t	32,039.56	2.62	98.67
mass_s	9,124.75	0.75	99.42
gravityForce_s	5,916.76	0.48	99.90
gravityForce_t	1,196.47	0.10	100.00
ground	0.00	0.00	100.00
displacement_s	0.00	0.00	100.00
displacement_t	0.00	0.00	100.00

Table 2: Response of components when model from Fig. 3 is given a step signal as input.

4.4 Example

A model from Fig. 3 is excited by two different signals, depicted in Fig. 4 and Fig. 5 respectively. For each experiment, components of the model are ranked with activity metrics (Fig. 1) and results are shown in Table 1 and 2 respectively.

As it can be seen, both rankings are very different, but so are the excitation signals. In the first example, the highest ranked components belong to the part of the model with slow dynamics, while in the second example is the part with faster dynamics much more excited and therefore also highly ranked. However, in the second example, simulation's communication interval is too large and thus there is a large error in weighting factors estimation.

5 Conclusion

Presentation of a model to user is an important aspect of modeling environments that helps with model understanding and maintenance. However, many Modelica's language features (e.g., inheritance) are important for effective implementation and to prevent code duplication, but may worsen the clarity of the model implementation. Furthermore, detailed models of complex systems are as hard to understand as reality. Therefore, we believe that there should be integrated a tool into the modeling environment which would help users, non-modeling specialist, to better understand the model and provide effective means for explaining the model behavior and model verification (and debugging). As it proposed in the paper, model order reduction and simplification techniques (e.g., ranking metrics used by these techniques) can be used for this purpose. It is important that the results are presented in the same form as the original model. Modelica's models are represented graphically, by object diagrams, or as a set of acausal differential-algebraic equations. Therefore, model order reduction and simplification techniques for both representation must be used. There are already many methods for simplifying (ranking) models represented with DAE system. We also showed that method for simplifying bond graphs (graphical modeling formalism) can be adapted to work with Modelica's object diagrams.

References

- [1] J. F. Broenik. Introduction to physical systems modeling with bond graphs. In *SiE whitebook on Simulation Methodologies*, pages 1–31, 1999.

- [2] Samuel Y. Chang, Christopher R. Carlson Carlson, and J. Christian Gerdes. A Lyapunov function approach to energy based model reduction. In *Proceedings of the ASME Dynamic Systems and Control Division – 2001 IMECE*, pages 363–370, New York, USA, 2001.
- [3] Modeling Chemical Reactions in Modelica By Use of Chemo-bonds. Cellier, f. e. and greifeneder, j. In *Proceedings of the 7th Modelica Conference*, pages 142–150, Como, Italy, 2009.
- [4] Sanjay Lall, Petr Krysl, et al. Structure-preserving model reduction for mechanical systems. *Physica D*, 284:304–318, 2003.
- [5] Loucas Sotiri Louca. *An Energy-based Model Reduction Methodology for Automated Modeling*. PhD thesis, University of Michigan, 1998.
- [6] Modelica Association. *Modelica Specification, version 3.2*, 2010. <http://www.modelica.org/documents/ModelicaSpec32.pdf>.
- [7] Modelica Association. *Modelica Standard Library 3.1, User's Guide*, 2010. <https://www.modelica.org/libraries/Modelica>.
- [8] Open Source Modelica Consortium. Openmodelica. <http://www.openmodelica.org>.
- [9] R. Rosenberg and T. Zhou. Power-based model insight. In *Proceedings of the ASME WAM Symposium on Automated Modeling for Design*, pages 61–67, New York, USA, 1988.
- [10] P. Schwarz et al. A tool-box approach to computer-aided generation of reduced-order models. In *Proceedings EUROSIM 2007*, Ljubljana, Slovenia, 2007.
- [11] R. Sommer, T. Halfmann, and J. Broz. Automated behavioral modeling and analytical model-order reduction by application of symbolic circuit analysis for multi-physical systems. In *Proceedings EUROSIM 2007*, Ljubljana, Slovenia, 2007.
- [12] Ralf Sommer, Thomas Halfmann, and Jochen Broz. Automated behavioral modeling and analytical model-order reduction by application of symbolic circuit analysis for multi-physical systems. *Simulation Modelling Practice and Theory*, 16:1024–1039, 2008.
- [13] Hubertus Tummescheit. *Design and Implementation of Object-Oriented Model Libraries using Modelica*. PhD thesis, Lund Institute of Technology, 2002.
- [14] T. Wichmann et al. On the simplification of nonlinear dae systems in analog circuit design. In *Proceedings of CASC'99*, pages 485–498, Munich, Germany, 1999.
- [15] Y. Ye and K. Youcef-Youmi. Model reduction in the physical domain. In *Proceedings of the American Control Conference*, pages 4486–4490, San Diego, CA, USA, 1999.

Simulation-Based Design of Aircraft Electrical Power Systems

Tolga Kurtoglu Peter Bunus Johan De Kleer

Palo Alto Research Center

3333 Coyote Hill Dr. Palo Alto, CA 94304

kurtoglu@parc.com

peter.bunus@parc.com

dekleer@parc.com

Rahul Rai

California State University

Department of Mechanical Engineering, Fresno, CA 93740

rarai@csufresno.edu

Abstract

Early stage design provides the greatest opportunities to explore design alternatives and perform trade studies before costly design decisions are made. The goal of this research is to develop a simulation-based framework that enables architectural analysis of complex systems during the conceptual design phase. Using this framework, design teams can systematically explore architectural design decisions during the early stage of system development prior to the selection of specific components. The analysis performed at this earliest stage of design facilitates the development of more robust and reliable system architectures. Application of the presented method to the design of a representative aerospace electrical power system (EPS) demonstrates these capabilities.

Keywords: simulation-based design; electrical power system; architectural design; concept generation

1 Introduction

The complexity of modern world engineered systems is growing constantly. New technologies are creating the potential for higher levels of integration and resulting systems contain a larger number of dynamically interacting components, relations among which are increasingly non-linear. This complexity, in turn, leads to unexpected behaviors and consequences, some of which have proven to be catastrophic. A key technical challenge in developing such complex sys-

tems is to ensure that the individual components and technologies are reliable, effective, and low cost, resulting in turn in safe, reliable, and affordable systems.

To address these challenges, DARPA's META Program is investing in novel methods for design and verification of complex systems. The META program is specifically aimed at compressing the product development and deployment timeline by enabling model-based design and manufacturing across the complex, heterogeneous, and physically-coupled electromechanical systems. Using this design paradigm, different "component model libraries" or "physics libraries" can be interchangeably used to instantiate a given system design such that a design can be analyzed and verified entirely independently of its physical manifestation [1].

On the other hand, ensuring safety, reliability, affordability, and performance requires the incorporation of subsystem and component functionality, decisions and knowledge into the product lifecycle as early as possible. Furthermore, formal tools and methodologies need to be in place to allow design teams to formulate a clear understanding of the impact of the decisions in the early design phases.

Developed as part of the META program, this paper presents a simulation-based design framework that enables architectural analysis of complex systems during the conceptual design phase. Using this framework, design teams can systematically explore architectural design decisions during the early stage of system development prior to the selection of spe-

cific components. The analysis performed at this earliest stage of design facilitates the development of more robust and reliable system architectures. In this paper, we describe the proposed framework and present the application of its use to the design of a representative aerospace electrical power system (EPS).

2 Integrated System Design and Analysis Framework

The framework is the basis for specifying system requirements, supporting design space exploration, and analyzing the performance associated with promising architectural design alternatives. To support a model-based design paradigm, the framework allows the designers to combine models from different domains into integrated system level models, and allow models of components and sub-systems to evolve throughout the design process. At the end, component models are composed into a system that achieves the intended functionality given specified requirements such as reliability, risk, and performance.

In what follows, we describe the constituent elements of the framework but first a brief overview of electrical power system design is provided.

2.1 Electrical Power System Design

An electrical power system is designed to deliver power to select loads, which in an aerospace vehicle would include subsystems such as the avionics, propulsion, life support, and thermal management systems. The EPS is required to provide basic functionality common to many aerospace applications: power storage, power distribution, and operation of loads [2].

An EPS system was originally designed by one of the co-authors using a failure-based design methodology at the early concept design phase [3]. Using this function-based design approach, several critical elements were identified and incorporated into the final design and realization of the system.

In the current realization of the system, which is illustrated in Fig. 1, the power storage consists of one or multiple battery modules, which are used to store energy for the operation of the loads. Any of the battery modules can be used to power any number of loads in the system. This requires the EPS to have basic redundancy and reconfiguration capability. Electromechanical relays or other electrical actuators can be used to route the power from the batteries to the loads. In addition, circuit breakers are added to the design at various points in the distribution net-

work to prevent overcurrents from causing unintended damage to the system components. Moreover, a sensor suite is designed in to allow monitoring of voltages, currents, temperatures, switch positions, etc. and to provide an integrated health management functionality. (More information on the existing electrical power system can be found in [2]).

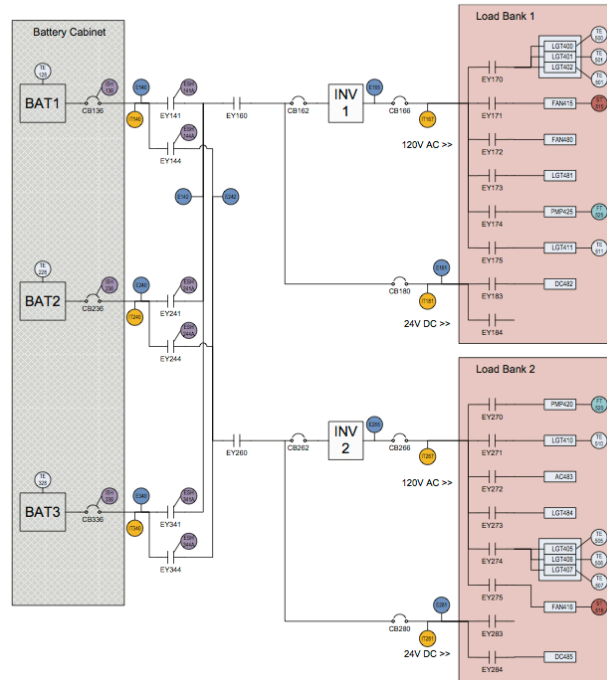


Fig. 1. The schematic of the existing electrical power system design architecture

2.2 A Modelica Library for an Aircraft EPS

In this paper, we extend our previous, function-based analysis of the EPS system and explore how well different EPS architectures meet specified reliability, risk, and performance requirements.

The building blocks within the presented model-based design environment are component objects consisting of a set of configurational, and behavioral models, component interfaces, and relationships between them. Accordingly, we build a Modelica-based [4] component model library, which will present the designers a set of available reconfigurable models, and include physical artifacts such as batteries, actuators, electrical switches, etc. The models are stored in an EPS Design Repository. For each component of the EPS Modelica library the nominal behavior was modeled and augmented with the relevant failure modes. A list of operating and possible failure modes of the EPS Modelica library components is depicted in Table 1.

Table 1. A list of components of the EPS Modelica libraries and the associated nominal operating and failure modes.

Model Element	Element Type	Operating and fault modes
Battery	Source	Nominal, AbruptParasiticLoad
CircuitBreaker	Electrical Circuit Breaker	Nominal, Tripped, FailedOpen, Stuck-Closed
Relay	Electromechanical Relay	NominalClosed, NominalOpen, StuckOpen
Inverter	Electrical Inverter	NominalOn, NominalOff, FailedOff
Temperature-Sensor	Temperature Sensor	Nominal, Drift, Offset, IntermittentOffset, Stuck
DCCurrent-Tranmitter	DC Current Transmitter (50A Max)	Nominal, Drift, Offset, IntermittentOffset, Stuck
DCVoltageSensor	DC Voltage Sensor 10HZ	Nominal, Drift, Offset, IntermittentOffset, Stuck
PositionSensor	Actuator Position Sensor 10HZ	Nominal, Stuck
ACResistor DCResistor	AC and DC Resistors	Nominal, FailedOff, IntermittentResistanceOffset, ResistanceDrift, ResistanceOffset
LargeFan	LargeFan	Nominal, OverSpeed, UnderSpeed, FailedOff
LightBulb	25W Light Bulb	Nominal, FailedOff
WaterPump	Water Pump	Nominal, FlowRestricted, FailedOff

For several components, models with different levels of detail have been created. For example, the *Inverter* component created Modelica models are ranging from very simple models that describe only the AC/DC power balance equation to models containing complicated electrical schematics including semiconductor components from the Electrical Standard Modelica Library. The reason for creating models of the same component with different levels of details was to compare how our proposed architecture analysis methods performs in very early stages of the conceptual analysis, when not so much details are available, to later stages when more details are added to the component models.

The nominal and the fault modes behavior of the Modelica EPS Library components have been validated by comparing the simulation behavior of two test models with measurements and sensor data from the Advanced Diagnostics and Prognostics testbed called ADAPT located at the NASA Ames Research Center [5,6]. (The ADAPT system consists of a controlled and monitored environment where faults can be injected into the system in a controlled manner and the performance of the test article is carefully monitored.)

The first test Modelica model, called the ADAPT Tier 1 (ADAPT Lite) model, depicted in Fig 2, contains a battery connected through a series of circuit breakers and relays to an inverter, and several loads consisting of a large fan, a DC resistor and AC resistor. The rotation speed of the fan is measured by a speed transmitter component. A series of four AC or DC voltage sensors and three current transmitters measure the voltage and current in different probing points of the circuit. The circuit breakers can be commanded externally to be closed or open and their position is monitored with the help of a position sensor connected to them.

The second EPS model (ADAPT Tier2) that has been tested and built in Modelica is depicted in Fig 3 and it is equivalent to the schematic represented in Fig 1. In this model the ADAPT Tier 2 EPS supplies power to five critical load functions and four non-critical loads distributed in two load banks. The battery cabinet unit contains three battery packs and several relays that control the connections between the load bank and the batteries. Similarly to the ADAPT Lite model, the testbed is controlled by a number of relays and monitored by a large set of sensors.

The ADAPT Tier 1 model has been validated against 39 experiments while the ADAPT Tie2 model has been validated against 33 experiments simulating nominal and faulty behavior of the EPS.

Since each component contains a description of the failure behavior besides the description of the nominal behavior, by systematically selecting a certain state of the system and inducing faults in the components, we were able to observe the effects of those faults on the system and automatically build a Failure Model and Effect Analysis (FMEA) table from the model.

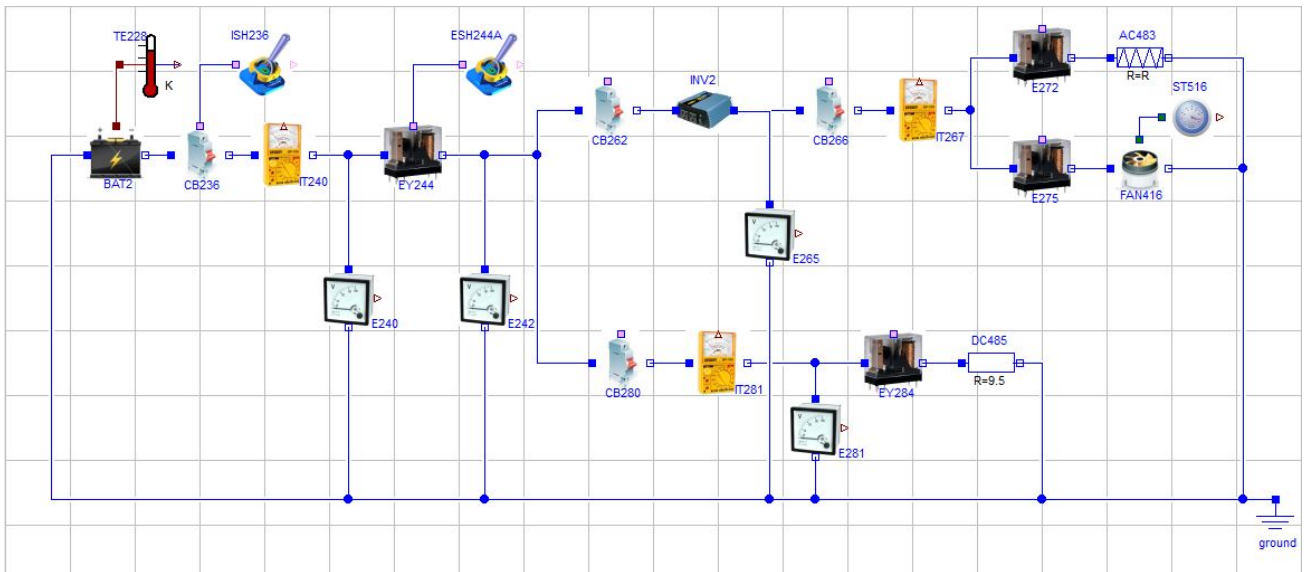


Fig. 2. The Modelica model of simple EPS system (ADAPT Tier 1).

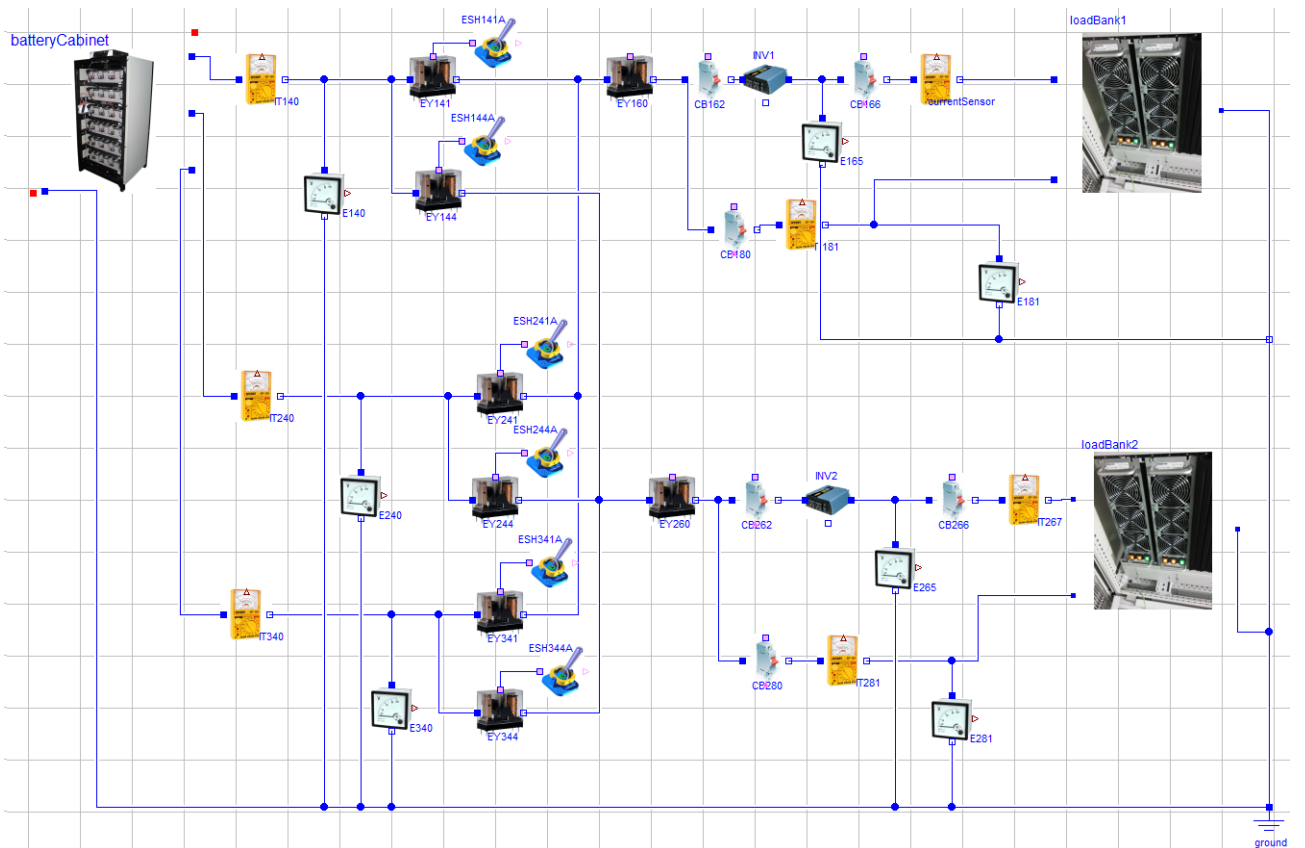


Fig. 3. The schematic of the existing electrical power system design architecture.

2.3 Design Space Exploration

This section examines the means of exploring the design space defined by combinations of generic EPS components. Feasible EPS candidate architectures are generated by a generative grammar based design space exploration technique. This generative

technique takes user specified EPS loads as input and satisfies system-level configuration requirements to generate feasible EPS candidate architectures. The component model library serves as the backbone of the proposed design space exploration approach. Using the models in the model library as building blocks, this generative graph grammar based technique configures “correct by construction” EPS ar-

b. A battery relay shall be installed in each battery circuit to enable the flight crew to isolate the battery from the rest of the electric subsystem. The battery relay shall be controlled by a crew station battery switch. Any circuits which must remain connected to the battery with the battery switch OFF shall be connected directly to the battery through suitable fuses or circuit breakers.

Fig. 4. An architectural design requirement that is used in derivation of a design grammar

chitectures that can be further studied by means of a simulation-based analysis.

Graph grammar based configuration approach uses graph as representation scheme. These approaches capture the transitions or the production rules for creating a solution, as opposed to storing the solutions themselves. Accordingly, a configuration's development from its inception to its final configuration is considered as a series of graph modifications. The initial specification can be represented as a simple graph in which the desired inputs and outputs are cast as arcs and nodes of the to-be-designed artifact. From this initial specification, the design process can be viewed as a progression of graph transformations that lead to the final configuration [7]. Recently, engineering design researchers have discovered that graph grammars provide a flexible yet ideally structured approach to the creation of complex engineering systems [8-10]. This interpretation of the design process makes graph grammars very suitable for computationally modeling the open-ended nature of conceptual design, where designers explore various ideas, decisions, and modifications to previous designs to arrive at feasible solutions.

Generating feasible EPS architecture using graph grammar based configuration approach is a two-step process: In the first step, we have developed an EPS system design grammar to encode design rules for constructing electrical power system architectures. For EPS, we have developed a 14-rule graph grammar that defines ways to generate feasible EPS architectures from multiple EPS requirement documents [11-15]. The rules are established prior to the design process and capture architectural design considerations that are inherent to the EPS design problem. One such EPS design requirement is shown in Figure

4. Similar design rules govern the mapping of functional requirements to components, or the physical compatibility between EPS components. Moreover, the graph grammar rules can be formulated in such a way that the final solution meets the constraints of the problem. The knowledge captured in the rules offer the option of exploring the design alternatives as well as automating the design generation process. Specifically, the developed design grammar encodes how specific system requirements can be embodied by selecting components from a full spectrum of electromechanical components represented in the component library.

In the second step, the graph transformation systems, or graph grammars, is invoked algebraically. Algebraic graph transformation methods rigorously define mathematical operations such as addition and intersection of graphs. A typical graph grammar rule is compromised of a left-hand side (LHS) and a right-hand side (RHS) (Figure 5). The LHS contains the conditions, upon which the applicability of a rule is determined. Accordingly, the LHS describes the state of the graph for a particular rule to be applicable. The RHS, on the other hand, contains the resulting graph transformation. It describes the new state of the graph after the application of the rule. By simply executing different combinations of grammar rules, a variety of feasible EPS architectures can easily be generated including the architecture of the ADAPT test bed shown in Figure 1.

A partial sequence of application of different EPS grammar rules to create a feasible EPS architecture is shown in Figure 6. In order to generate a feasible EPS architecture the approach starts with a seed graph. The seed graph for EPS design space exploration is graph based representation of three main

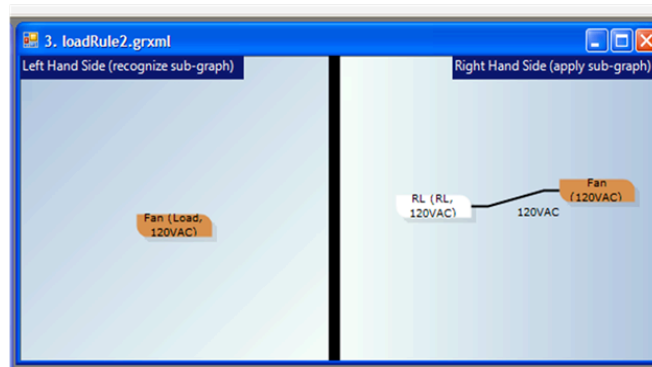


Figure 5. A graph grammar rule for EPS architecture generation

2.4 Simulation-Based Performance Analysis

Let us consider the following set of safety, functional and performance requirements imposed on the EPS system.

Safety Requirements imposed in the architecture:

- “AFGS-87219A: A battery relay shall be installed in each battery circuit to enable the flight crew to isolate the battery from the rest of the electric subsystem.”
- “MIL-STD 7080: A switch or relay shall be connected in series with the circuit breaker when a switching capability is required for a circuit protected by a circuit breaker.”

Functional Requirements:

- “MIL-STD 704F: “Loads should not introduce excessive current distortion such that other EPS functionality is effected.”

Performance Requirements:

- “MIL-STD-1275D: The [28 VDC electrical power system] circuit steady-state voltage shall be between 25 VDC and 30 VDC.”
- “MIL-STD-1275D: The rotational speed of cooling fan system should be between 765 and 900 rpm.

As it was described in Section 2.3 feasible EPS candidate architectures are generated by a generative grammar based design space exploration technique. The graph grammar configuration approach is able to impose the safety requirements detailed above by encoding the safety requirements in graph grammar rules that are applied by the transformation system resulting in an architecture that is correct by definition. The Modelica model of a simple EPS system, (ADAPT Tier2) depicted in Fig 2 satisfy both safety requirements: the relay EY244 will isolate the battery from the rest of the electrical circuit (the first safety requirement AFGS-87219A) while the circuit breaker-relay pairs (CB236-EY244, CB266-E272, CB266-EY275, CB280-EY284) will satisfy the second requirement from MIL-STD 7080.

The functional and performance requirements, on the other hand, are verified by simulation. Simulation-based design methods require the capability of specifying detailed input design parameters and using them to obtain a model response. Accordingly, we use a simulation process which allows system designers use to account for the effects of variability in the input and design parameters on the model response, thereby incorporating uncertainty into the design process. In this research, we use a sampling

based technique to perform a simulation-based analysis of system performance. This analysis provides a means to estimate the probability of system response and assess how well a candidate system design meets its requirements.

For example, in the ADAPT Tier 1 EPS the designer has the choice of using a Xantrex Prosine 1000 Inverter or a Xantrex Freedom HW 1000 Inverter. Both variants will satisfy the safety requirements imposed on the architecture. The Xatrex Prosine 1000 Watt Inverter has a peak efficiency of 90% while the output voltage (over full load and battery voltage range) is around 120 Vac - 10 %/+4 %. The range of the output voltage for this type of inverter can be defined as a triangular probability distribution function. The output voltage histogram for 200 samples is depicted in Figure 8.

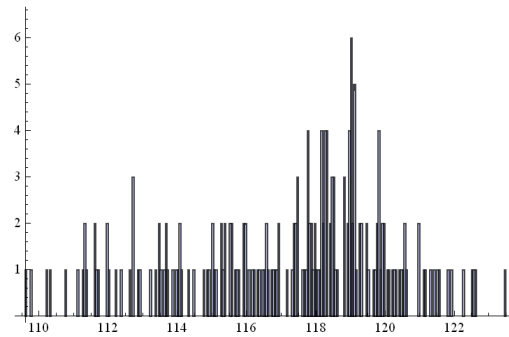


Figure 8. The Xantrex Prosine 1000 Inverter output voltage histogram.

We perform a simulation based performance analysis and we compute the rotational speed of the cooling fan for different output voltages of the inverter. The histogram of the rotational speed of the cooling fan shows that using a Xantrex Prosine 1000 Inverter is a valid architecture, which satisfies the performance requirement that the speed of the cooling fan should be between 765 and 900 rpm.

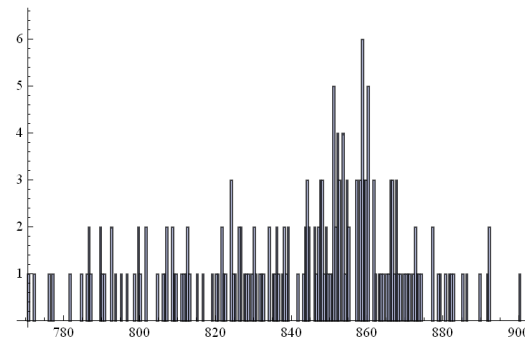


Figure 9. The histogram of the rotational speed of the cooling fan when a Xantrex Prosine 1000 inverter is used.

The Xantrex Freedom HW 1000 Invertor has slightly different characteristics: a peak efficiency of 83% and an output voltage (over full load and battery voltage range) around 115 Vac +/-10 Vac that can be also approximated as a triangular distribution function. The histogram of the output voltage is shown in Figure 10.

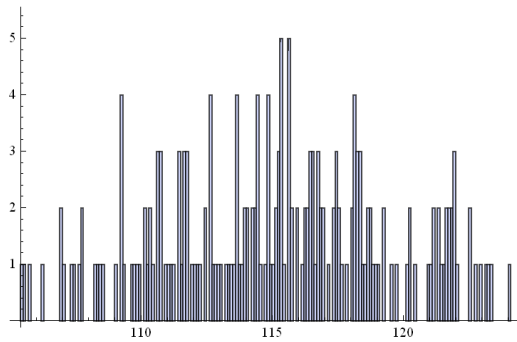


Figure 10. The Xantrex Freedom HW 1000 Invertor output voltage histogram.

A performance based simulation shows that the rotational speed of the cooling fan can sometimes drop below 765 rpm for certain AC output voltages of the Xantrex Freedom HW 1000 Invertor (see Figure 11).

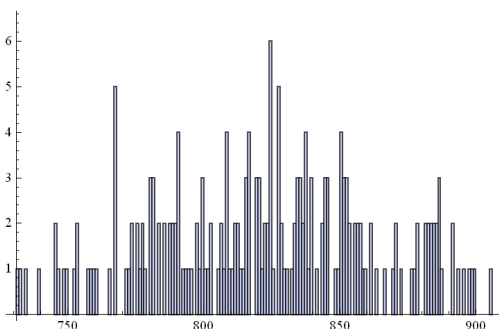


Figure 11. The histogram of the rotational speed of the cooling fan when a Xantrex Freedom HW 1000 invertor is used.

Since the performance requirements of an EPS architecture using a Xantrex Freedom 100 Invertor are not met, this architecture can be discarded from the list of alternative EPS designs.

3 Conclusions

We have outlined a framework for simulation-based design that integrates architectural synthesis and analysis of complex systems during the conceptual design phase. The incorporation of automated design space exploration methods with Modelica broadens the scope of the capabilities of the language, and enables it to support architectural trade studies before

costly design decisions are made. In this paper, we presented preliminary results of our study. In the future, we plan to fully integrate and automate the architectural synthesis and analysis approaches described in this paper.

References

- [1] Defense Advanced Research Projects Agency (DARPA), Tactical Technology Office (TTO) META-II, BAA-10-59, 2010.
- [2] S. Poll, A. Patterson-Hine, J. Camisa, D. Garcia, and D. Hall, "Advanced Diagnostics and Prognostics Testbed," in *18th International Workshop on Principles of Diagnosis (DX-07)* Nashville, TN, 2007.
- [3] T. Kurtoglu, Jensen, D., Tumer I.Y., "A Functional Failure Reasoning Methodology for Evaluation of Conceptual System Architectures", *Journal of Research in Engineering Design*, published online, January 31, 2010.
- [4] Modelica Language, www.modelica.org
- [5] Poll Scott, Ann Patterson-Hine, Joe Camisa, David Garcia, David Hall, Charles Lee, Ole J. Mengshoel, Christian Neukom, David Nishikawa, John Ossenfort, Adam Sweet, Serge Yentus, Indranil Roychoudhury, Matthew Daigle, Gautam Biswas, and Xenofon Koutsoukos. (2007). "Advanced Diagnostics and Prognostics Testbed." In *Proceedings of the International Workshop on Principles of Diagnosis (DX-07)*. (Nashville, TN, May 2007, 2007).
- [6] NASA Ames Research Center (2006) "Advanced Diagnostics and Prognostics Testbed (ADAPT) System Description, Operations, and Safety Manual," February, 2006.
- [7] Cagan, J., 2001, "Engineering Shape Grammars," *Formal Engineering Design Synthesis*, Antonsson, E. K., and J. Cagan, eds., Cambridge University Press.
- [8] Rai, R., Kurtoglu, T., and Campbell, M., 2009, "Stochastic interactive graph grammar search for conceptual design" *ASME Journal of Computing and Information Sciences in Engineering* (Accepted for Publication with review).
- [9] Kurtoglu, T., Campbell, M., "Automated Synthesis of Electromechanical Design Configurations from Empirical Analysis of Function to Form Mapping". *Journal of Engineering Design*, Vol. 20 (1), Feb 2009.

- [10] Shea, K., J. Cagan, and S.J. Fenves, 1997, "A Shape Annealing Approach to Optimal Truss Design with Dynamic Grouping of Members", ASME Journal of Mechanical Design, Vol 119, No. 3, pp. 388-394.
- [11] DEPARTMENT OF DEFENSE, "Aircraft electric power characteristics", MIL-STD-704F, 12 March 2004.
- [12] DEPARTMENT OF DEFENSE, "Air Force Specification Guide: Electrical Power Systems", Aerospace Vehicles, AFGS-87219A, 30 March 1993.
- [13] DEPARTMENT OF DEFENSE, "Characteristics of 28 Volt DC Electrical Systems in Military Vehicles", MIL-STD-1275D, 29 August 2006.
- [14] DEPARTMENT OF DEFENSE, "Selection and Installation of Aircraft Electronic Equipment", MIL-STD-7080, 31 May 1994
- [15] DEPARTMENT OF DEFENSE, "Joint services specification guide (JSSG-2009) air vehicle subsystems", Appendix H, 30 October 1998.

HumMod - Large Scale Physiological Models in Modelica

Jiri Kofranek Marek Matejak Pavol Privitzer

Institute of Pathophysiology, First Faculty of Medicine, Charles University

U nemocnice 5, 128 53 Praha 2, Czech Republic

kofranek@gmail.com matejak.marek@gmail.com pavol.privitzer@lf1.cuni.cz

Abstract

Modelica is being used more and more in industrial applications, but Modelica is still not used as much in biomedical applications. For a long time we have mostly been using Matlab/Simulink models, made by Mathworks, for the development of models of physiological systems. Recently, we have been using a simulation environment based on the Modelica language. In this language, we implemented a large scale model of interconnected physiological subsystems containing thousands of variables. Model is a richly hierarchically structured, easily modifiable, and “self-documenting”. Modelica allows a much clearer than other simulation environments, to express the physiological nature of the modeled reality.

Keywords: simulation; physiology; large-scale model

1 Introduction

It is simply amazing how fast the Modelica simulation language adopted various commercial development environments. Modelica is being used more and more in industrial applications. However, Modelica is still not used as much in biomedical applications.

The vast majority of biomedical simulation applications are still done in casual, block-oriented environments. These include referencing database development environments for biomedical models (such as the JSIM language - <http://physiome.org/model/doku.php> or CellML language - <http://www.cellml.org/>).

A frequently used environment in biology and medicine is Matlab/Simulink – monographs dedicated to biomedicine models are usually equipped with additional software used in this environment, but so far without the use of new acasual or non-casual Simulink libraries, such as [24, 28, 32].

However, already in 2006, Cellier and Nebot [5] pointed out the benefits of Modelica, when used for clear implementation of physiological systems descriptions and interpretations. The classic McLeod’s circulation system model was implemented by PHYSBE (PHYS-

iological Simulation Benchmark Experiment) [25, 26, 27]. The difference is clearly seen, if we compare the Cellier model implementation [5] with the freely downloadable version of the PHYSBE model implementation in Simulink <http://www.mathworks.com/products/demos/simulink/physbe/>.

Haas and Burnhan, in their recently published monograph, pointed out the benefits and large potential of the Modelica language used for modeling medically adaptive regulatory systems [9]. The most recent, Brugård [4] talks about work on the implementation of the SBML language (<http://sbml.org/>) in the Modelica language. This would enable us in the future, to simply run models, whose structure is described in the SBML language, on development platforms, based on the Modelica language.

2 Web of physiological regulations

Thirty-nine years ago, in 1972 Guyton, Coleman and Granger published an article in the Annual Review of Physiology [9] which at a glance was entirely different from the usual physiological articles of that time. It was introduced by a large diagram on an insertion. Full of lines and interconnected elements, the drawing vaguely resembled an electrical wiring diagram at first sight (Fig. 1). However, instead of vacuum tubes or other electrical components, it showed interconnected computation blocks (multipliers, dividers, adders, integrators and functional blocks) that symbolized mathematical operations performed on physiological variables. In this entirely new manner, using graphically represented mathematical symbols; the authors described the physiological regulations of the circulatory system and its broader physiological relations and links with the other subsystems in the body – the kidneys, volumetric and electrolyte balance control, etc. Instead of an extensive set of mathematical equations, the article used a graphical representation of mathematical relations. This syntax allowed depicting relations between individual physiological variables graphically in the form of interconnected blocks representing mathematical operations. The whole dia-

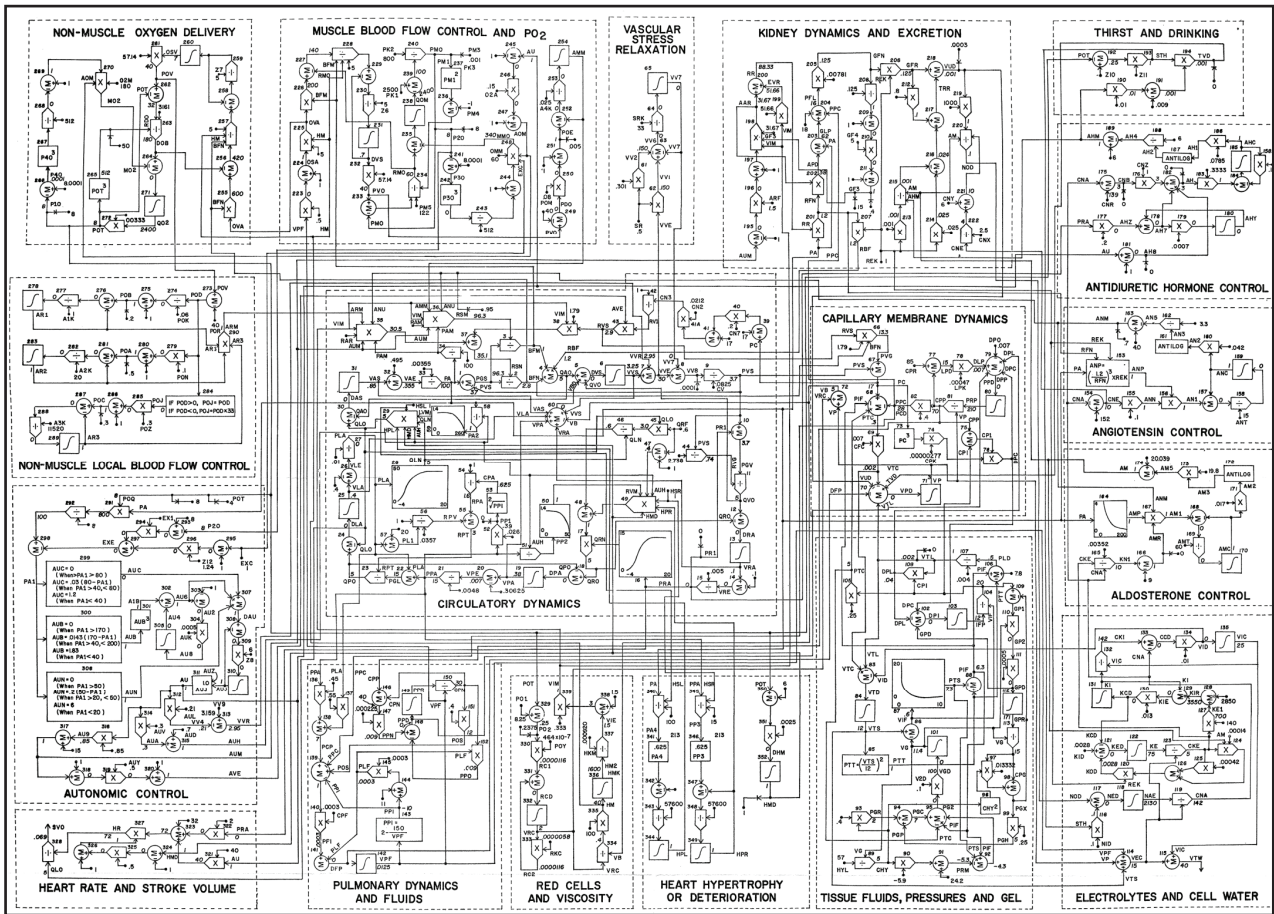


Figure 1: Guyton's blood circulation regulation diagram from 1972.

gram thus featured a formalized description of physiological relations in the circulatory system using a graphically represented mathematical model.

Guyton's model was the first extensive mathematical description of the physiological functions of interconnected body subsystems and launched the field of physiological research that is sometimes described as "integrative physiology" today. Just as theoretical physics tries to describe physical reality and explain the results of experimental research using formal means, "integrative physiology" strives to create a formalized description of the interconnection of physiological controls based on experimental results and explain their function in the development of various diseases.

From this point of view, Guyton's model was a milestone, trying to adopt a systematic view of physiological controls to capture the dynamics of relations between the regulation of the circulation, kidneys, the respiration and the volume and ionic composition of body fluids by means of a graphically represented network.

Guyton's graphical notation was soon adopted by

other authors – such as Ikeda et al. (1979) in Japan [13]

and Amosov et al. (1977) in the former USSR [2]. However, the graphical notation of the mathematical model using a network of interconnected blocks was only a graphical representation – Guyton's model and later modifications (as well as the models of other authors that adopted Guyton's representative notation) were originally implemented in Fortran and later in C++.

Today the situation is different.

Now, there are specialized software simulation environments available for the development, debugging and verification of simulation models, which allow creating a model in graphical form and then testing its behavior. One of these is the Matlab/Simulink development environment by Mathworks, which allows building a simulation model gradually from individual components – types of software simulation elements that are interconnected using a computer mouse to form simulation networks.

Simulink blocks are very similar to the elements used by Guyton for the formalized representation of

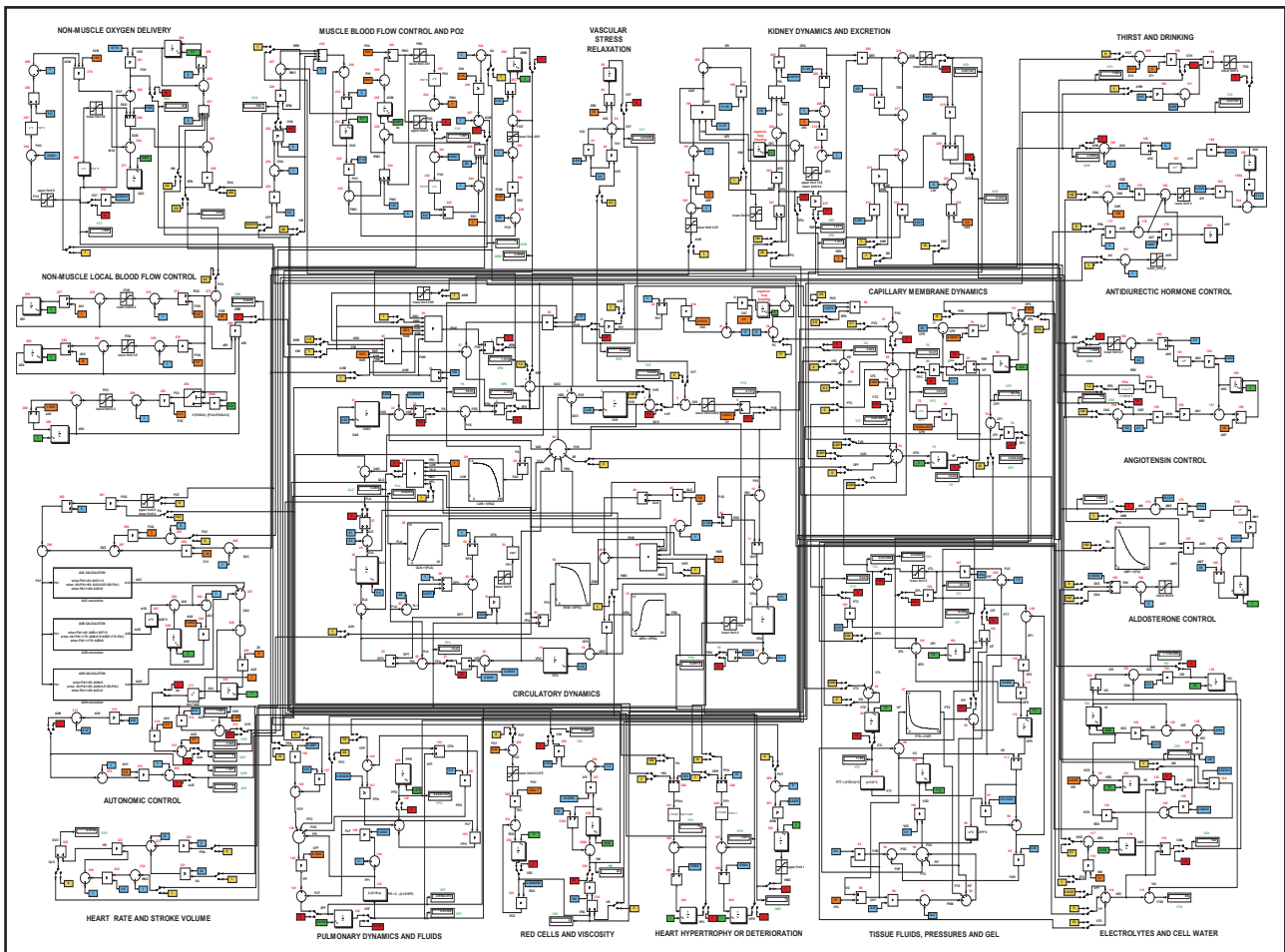


Figure 2: The implementation of Guyton’s model in Simulink preserves the original arrangement of elements in Guyton’s graphic diagram.

physiological relations. The only difference is in their graphical form. This similarity inspired us to use Simulink to revive Guyton’s good, classic diagram and transform it into a working simulation model. When implementing the model in Simulink, we used switches that allow us to connect and disconnect individual subsystems and control loops while the model is running. We strove to keep the appearance of the Simulink model identical to the original graphic diagram – the arrangement, wire location, variable names and block numbers are the same.

The simulation visualization of the old diagram was not without difficulties – there are errors in the original graphic diagram of the model! It does not matter in the hand-drawn illustration but if we try to bring it to life in Simulink, the model as a whole collapses immediately. A detailed description of the errors and their corrections is in [23].

Our Simulink implementation of Guyton’s (corrected) model (Figs. 2 and 3) is available for download at www.physiome.cz/guyton. Also available at that address is our Simulink implementation of a much more complex, later model from Guyton et al. There is also

a very detailed description of all applied mathematical relations with an explanation.

3 Block-oriented simulation networks for physiology

Block-oriented simulation languages, of which Simulink is a typical example, allow assembling computer models from individual blocks with defined inputs and outputs. The blocks are grouped in libraries; when building a model, a computer mouse is used to create individual block instances, with inputs and outputs connected through wires that “conduct” information.

A Simulink network can be arranged hierarchically. Blocks can be grouped into subsystems that communicate with their ambient environment through defined input and output “pins”, making “simulation chips” of a sort. A simulation chip hides the simulation network structure from the user, much like an electronic chip hiding the interconnection of transistors and other electronic elements. Then the user can be concerned

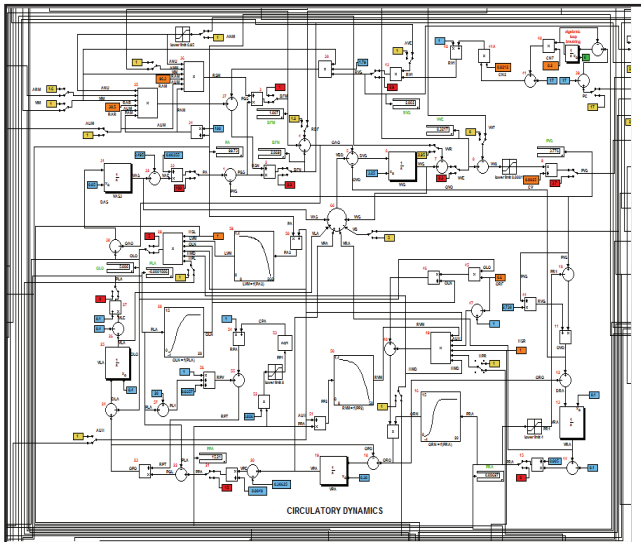


Figure 3: Circulatory dynamics - more detailed central structures of the Simulink implementation of Guyton's model, representing flows through aggregated parts of the circulatory system and the activity of the heart as a pump.

just with the behavior of the chip and does not have to bother about the internal structure and calculation algorithm.

The behavior of a simulation chip can be tested by monitoring its outputs using virtual displays or virtual oscilloscopes connected to it. This is very useful especially for testing the behaviour of a model and expressing the mutual relations of variables.

Simulation chips can be stored in libraries and users can create their instances for use in their models. For example, we created a Physiobrary for modelling physiological regulations.

Hierarchical, block-oriented simulation tools are thus used advantageously in the description of the complex regulation systems that we have in physiology.

A formalized description of physiological systems is the subject matter of PHYSIOME, an international project that is a successor to the GENOME project. The output of the GENOME project was a detailed description of the human genome; the goal of the PHYSIOME project is a formalized description of physiological functions. It uses computer models as its methodological tool [3, 12].

Several block-oriented simulation tools developed under the PHYSIOME project have been used as a reference database for a formalized description of the structure of complex physiological models. These include JSIM (<http://www.physiome.org/model/doku.php>) and CellML (<http://www.cellml.org>).

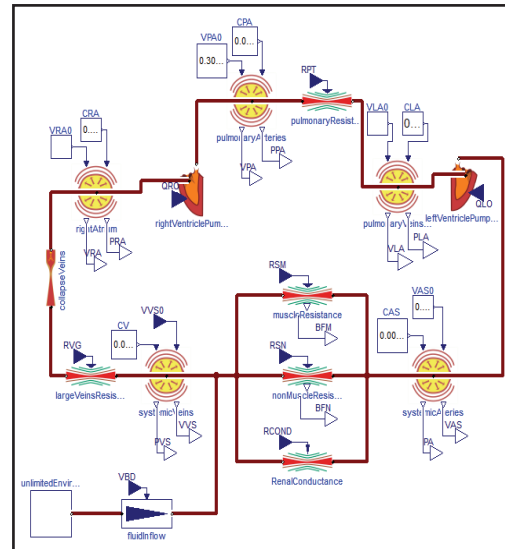


Figure 4: The same model structure as is shown in figure 3 implemented in Modelica. The structure of the model in Simulink corresponds to the structure of computational steps, while the Structure of Modelica model reflects the structure of the modeled physiological reality.

4 From Simulink to Modelica in modeling of large-scale physiological systems

We have been using Matlab and Simulink for years to create and develop models of physiological systems [16, 17, 23] and have also been developing the relevant application Simulink library – the Physiobrary (<http://www.physiome.cz/simchips>).

We have also developed the relevant software tools that simplify the transfer of models implemented in Simulink over to development environments (ControlWeb and Microsoft .NET), where we create tutorial and education simulators [18, 22]. Our development team gained invaluable experience in previous years working with the Matlab/Simulink development environment made by the renown company MathWorks. On the other hand, we were also attracted by the acasual development environments using the Modelica language.

In the Modelica language environment the essence of physiological regulation is much clearer than in Simulink causal network (see Figures 3 and 4). We were facing a decision whether to continue with the development process of physiological system models in Simulink (using new acasual libraries), or to make a radical decision and switch to the new Modelica language platform.

Our decision was affected by our efforts to imple-

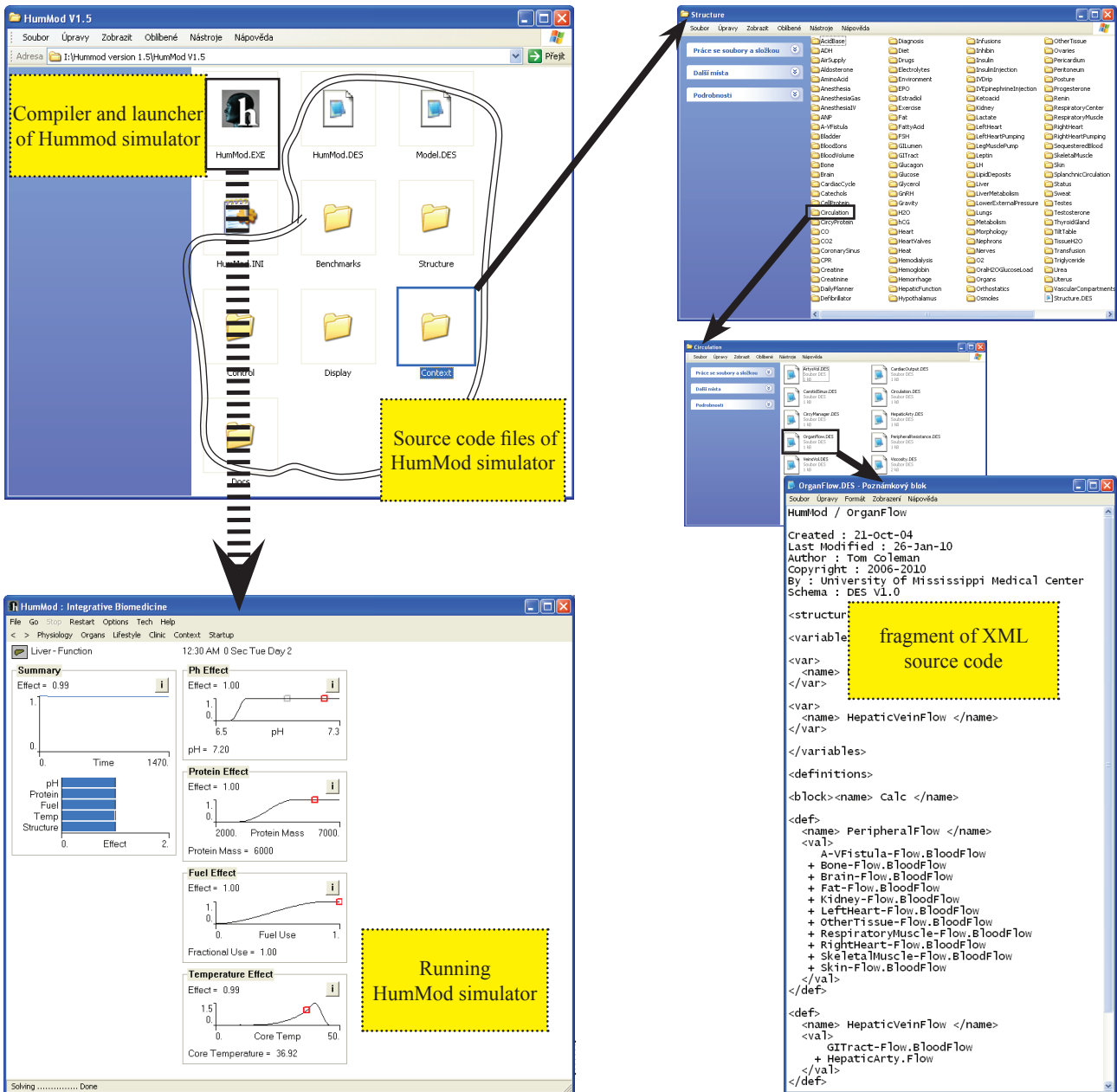


Figure 5: All necessary files of the Quantitative Human Physiology tutorial simulator (called the HumMod by the authors in the last version). This simulator has been designed for the Windows operating system and does not require special installation. Only zip files must be unzipped into a selected folder. After you click the Hummod.exe icon, the translator translates the source text embedded within hundreds of directories and more than two thousand files and initiate its own simulator. Even though the source text of the simulator and the entire mathematical model on the background is offered as an open source (and in theory, the user may modify the model), the navigation through thousands of mathematical relations and viewing thousands of XML and interconnected files is rather difficult.

ment a large model made by Guyton's disciples and followers. Their *Quantitative Human Physiology* model is an extension of a tutorial simulator called the *Quantitative Circulatory Physiology (QCP)* [1]. *Quantitative Human Physiology (QHP)* simulator [10], which is now distributed as "*HumMod*" [11], represents today's most comprehensive and largest model of physiological functions.

The *HumMod* model contains more than 4000 variables and at the present time, it probably represents the largest and most extensive model of physiological regulations. It enables the user to simulate a wide range of pathological stages and statuses, including the effects of the relevant applied therapy. The authors developed a special block-oriented simulation system to represent the complex model structure. Compared

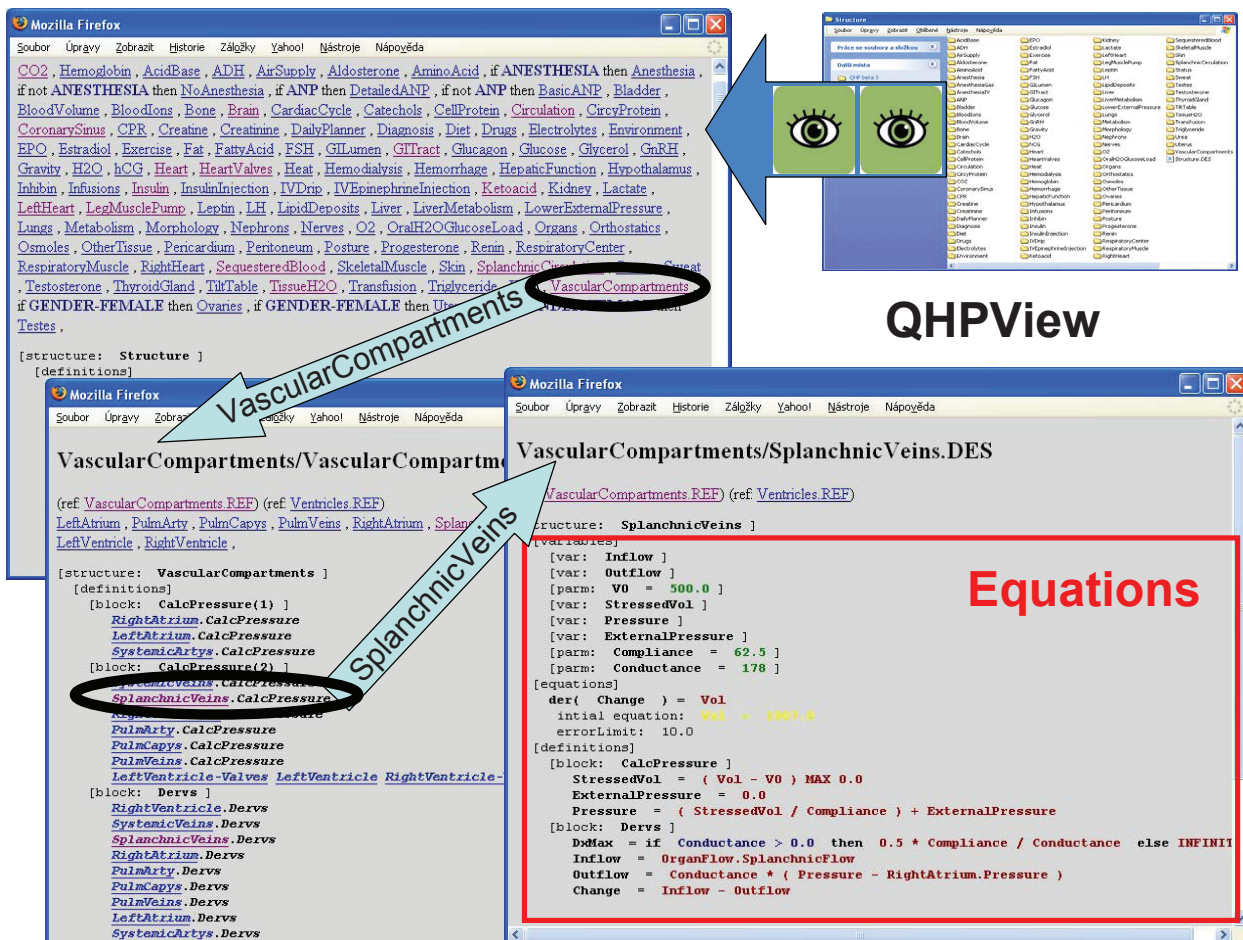


Figure 6: Visualization tool QHPView, created by us, simplifies viewing of the QHP/HumMod simulator structure, containing more than two thousand XML files, scattered in thousands of directories, where quotations and links between them may not be apparent.

with the previous *QCP* simulator, whose mathematical background is hidden from the user in its source code written in C++, the *HumMod* simulator uses a different approach. The HumMod authors decided to separate the simulator implementation and description of the model quotations, in order to make the structure of the model more clear and apparent for the larger scientific community.

In 1985 the architect of this model, Thomas Coleman, had already created a special language used to write the model structure, as well as the element definitions into the simulator user interface. The language is based on modified XML notation. The model is then written by using XML files. A special converter/decoder (DESolver) converts XML files into executable simulator code.

A detailed description of this language and DESolver converter, as well as the relevant educational tutorial, is freely accessible on the web page of the University of Mississippi (<http://physiology.umc.edu/themode-lingworkshop>). The new HumMod model is written

in the XML language as well. Its structure with all details may be found at (<http://HumMod.org>), published as an open source.

Therefore, the user can modify this model as he wishes. However, the model description has been divided into more than three thousand XML files in more than thousand directories, from which the special solver creates and executes the simulator (Figure 5).

The entire structure of the model and following links and references are not easily identifiable. That is why the international research and development team in its SAPHIR project (System Approach for Physiological Integration of Renal, cardiac and respiratory control) decided to use the old Guyton models from 1972 [9] and the Ikeda model from 1979 [13] for the creation of its new and extensive model of physiological functions instead of the freely available *QHP* model. The source codes of the *QHP* model appeared unclear or hard-to-understand to those involved in this project [31].

We have been able to create a special software tool called *QHPView* (Figure 6), which is able to create a clear and legible overview of mathematical relations and connections from thousands of source codes. We are offering this tool as an open source on the web page at (<http://physiome.cz/HumMod>). First, we tried to implement the *QHP/HumMod* model in the Simulink environment.

The model contains a wide range of relations that offer solutions for implicit equations. That is why the implementation of this block-oriented model (outputs from one block are used as inputs for the next blocks) is very difficult and as the implementation got more and more complex, the transparency of this model went down quickly. The use of new acasual Simulink libraries in this complex model proved to be problematic and the transparency of the model improved only a little bit.

Therefore, we decided to stop using the Simulink implementation and began to implement in Modelica language (using the Dymola environment). Very quickly we discovered that the *implementation of a large and extensive model in Modelica is much more effective than using acasual libraries in Simulink*. When we compared the Simulink and Modelica implementations we also discovered a significant difference. Mainly due to the fact that the new acasual libraries are only acasual superstructure of Simulink and not an objectively oriented modeling language based on equations, as the Modelica language is.

Therefore, if we compare the development environments based on the simulation language Modelica with the Matlab/Simulink development environments made by Mathworks, we may say the following:

- contrary to Simulink, the model implemented in Modelica much better reflects the essentials and base of the modeled reality and the *simulation models are more clear, readable and less error prone*;
- the *object architecture* in Modelica enables the user to build and tweak models with an hierarchical structure gradually, while using *reusable element libraries*;
- contrary to Simulink (which is the industrial standard from Mathworks), Modelica is a *non proprietary programming language* and therefore, it may contain various commercial and non-commercial developing environments competing between each other. This language is used for specific problem solutions originating in various application fields (for commercial and non-co-

mmercial specialized libraries);

- in Modelica it is possible to *combine causal (mostly signals) and acasual links*; and unlike in Simulink, it is also possible, (within interconnected blocks) to create algebraic loops - Modelica compiler uses symbolic manipulations to resolve the loops automatically (when possible) and therefore *the disconnection of algebraic loops is the task for the development environment and not for the programmer*.

The above specified reasons led us to use, as the main implementation tool for the model creation, the Modelica language and we also gradually stopped using the Matlab/Simulink environment [20].

5 HumMod in Modelica

The implementation of the *HumMod* model clearly shows the benefits of the model creation process when done in the Modelica language. If we compare the complex structure of the *HumMod* model by using the visualization option in *QHPView* (Figure 5) with examples of implementations done in the simulation language Modelica, shown in Figures 7-13, we can see that the acasual implementation done in Modelica creates a transparent and legible model structure and

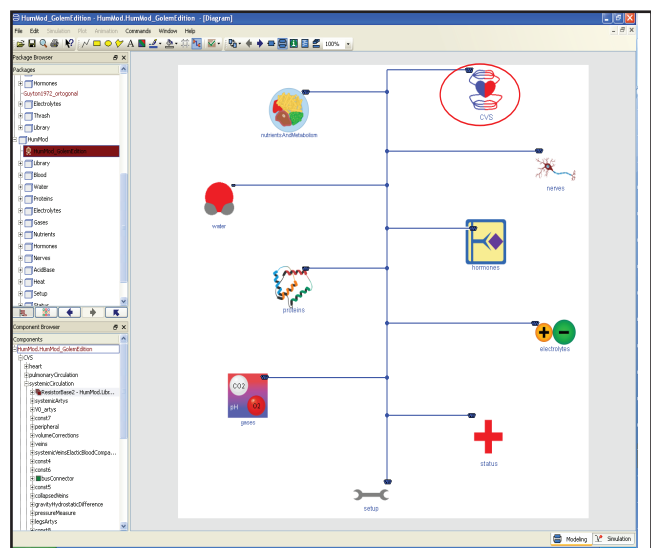


Figure 7: Structure of Hummod model. Model consist of cardiovascular component (CVS), nutrient and metabolism component, water and osmolarity component, proteins component, O₂, CO₂ and acid-base regulation component, electrolyte component, nervous regulation component, hormone regulation component, status of virtual pateint component and setup component. All components ar connected with bus connectors.

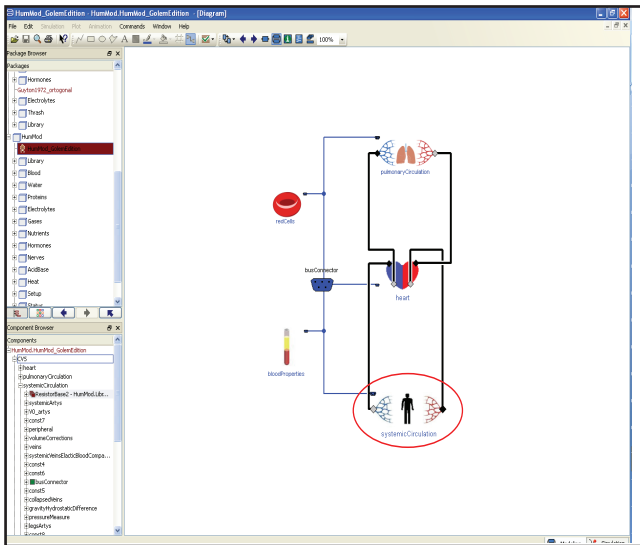


Figure 8: Structure of cardiovascular component (CVS class).

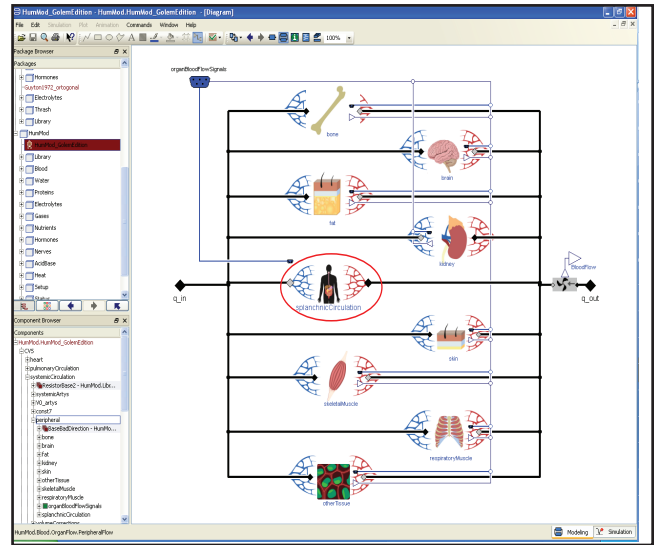


Figure 10: Structure of systemic peripheral circulation component (Peripheral class).

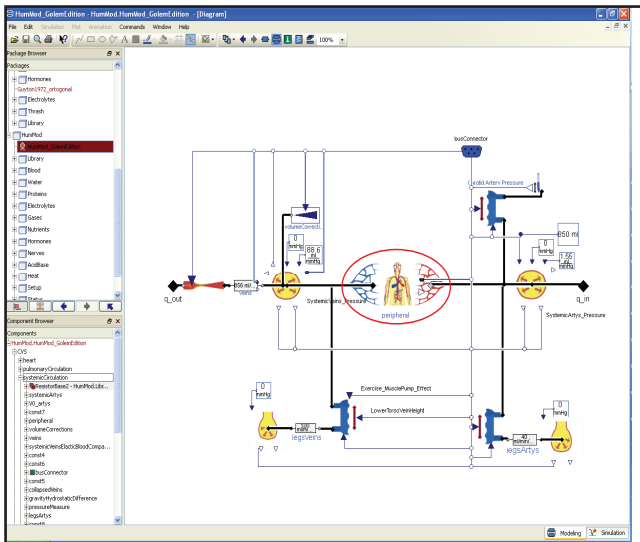


Figure 9: Structure of systemic circulation component (SystemicCirculation class).

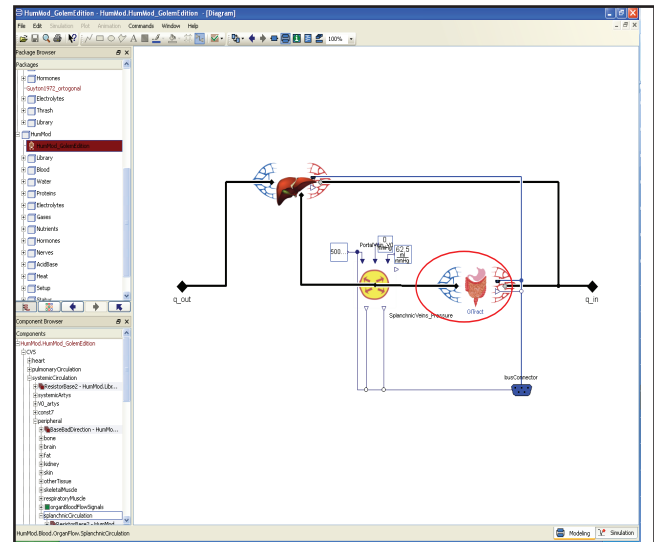


Figure 11: Structure of splanchnic circulation component (SplanchnicCirculation class).

therefore offers easier model modifications.

The *HumMod* model implemented in Modelica is being currently *modified and extended*.

Modifications and extensions of *HumMod* were partially taken from our original model Golem [16, 17] and further modified according to newest findings and experiences. Our modifications are mainly extensions, which improve the usability of the model during the modeling of difficult disorders in acid-base, ionic, volume and osmotic homeostasis of inner environments, which is very important for urgent medicinal statuses. Our modification of the HumMod model is based mainly on the process of *re-modeling the subsystem of acid-base balance*, which is based in the original *QHP* on the so-called Stewart acid-base balance theo-

ry. Simply put, the so-called “modern approach“ of Stewart [30] and his followers (e.g. Fencl et al. [8], Sirker et al. [29]) explaining disorders in the acid-base balance, uses mathematical relations calculating the concentration of hydrogen ions $[H^+]$ from partial pressure CO_2 in plasma (pCO_2), total concentration ($[Buf_{tot}]$), weak (partially dissociated) acids ($[HBuf]$) and their base ($[Buf]$), where:

$$[Buf_{tot}] = [Buf] + [HBuf]$$

and from the difference between the concentration of fully dissociated cations and fully dissociated anions in *SID* (strong ion difference):

$$[H^+] = Function(pCO_2, SID, Buf_{tot})$$

The problem of this approach is that the precision of acid-base calculations in the model depends on the

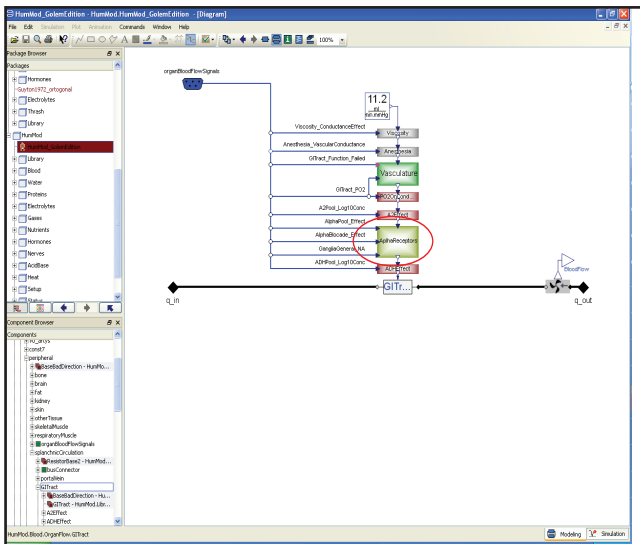


Figure 12: Structure of gastrointestinal vascular resistance component (GITract class).

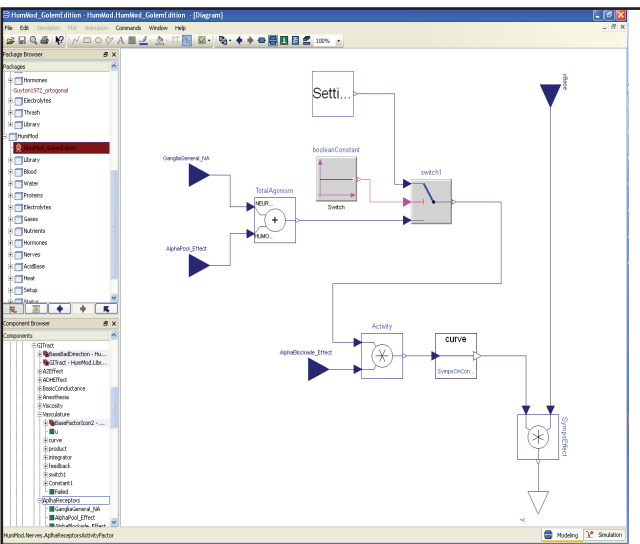


Figure 13: Structure of component calculating influence of alpha receptors stimulation on gastrointestinal vascular resistance (AlphaReceptors class).

precision of the SID calculation, that is the difference between the concentration of fully dissociated cations (that is mainly sodium and potassium) and fully dissociated anions (mostly chlorides). Imprecision that is created during the modeling of sodium, potassium and chlorides intake and excretion are transferred and reflected by the imprecision in the modeling process of the acid-base status.

Even though Hester et al. [11], significantly improved the modeling of reception and excretion of sodium, potassium and chlorides in kidneys in his *HumMod* model, if we model a long-term status (when nothing is happening with the virtual patient), the virtual patient (in the current model version) has a tendency to fall into slight and steady metabolic acidosis after one

month of the simulated time.

Our evaluative approach towards the modeling and evaluation of disorders in acid-basic balance [14, 15, 21] is based on the modeling and evaluation of two flows – the creation and excretion of CO₂ and the creation and excretion of strong acids, connected through the purification systems of each part of the bodily fluids. This approach, according to our opinion, better explains the physiological causality of acid-base regulations, rather than direct modeling of acid-base disorders through the balancing of accompanying electrolytes. Besides, the fidelity and truthfulness of the modeling process is getting better; mainly in mixed (acid-base and electrolyte) disorders in inner environments.

Another important modification of the *HumMod*, is the fact that the *model was extended by adding the dependency of the potassium flow on the intake of glucose as a result of insulin*, which enables us to model (besides other things), the influence of potassium solution infusions with insulin and glucoses, which are distributed in acute medicine for treating potassium depletions.

We have been using this “balancing and evaluation” approach [18] towards the modeling of acid-base balance in our old “Golem” simulator [17]. The extended *HumMod* model serves as the base for the educational simulator „*eGolem*“, used in medical tutoring in clinical physiology of urgent statuses which is being currently developed.

On the webpage <http://physiome.cz/HumMod> you may find the updated and current structure of our implementation of the *HumMod* model („*HumMod-Golem edition*“). In collaboration with M. Tiller we are preparing a detailed description of this model with extensive descriptions of the various physiological regulatory circuits.

6 From a model to the simulator

A simulation model, implemented in a sophisticated development environment, cannot be used for education as is alone. It is the implementation of the formalized description of the modeled reality that enables testing of the behavior of the mathematic model during various input values and the search for model quotations and parameters, which within the established precision range, can ensure the sufficient compatibility of the behavior of the model with the modeled system (model identification).

Even after this goal is reached, there is still a long

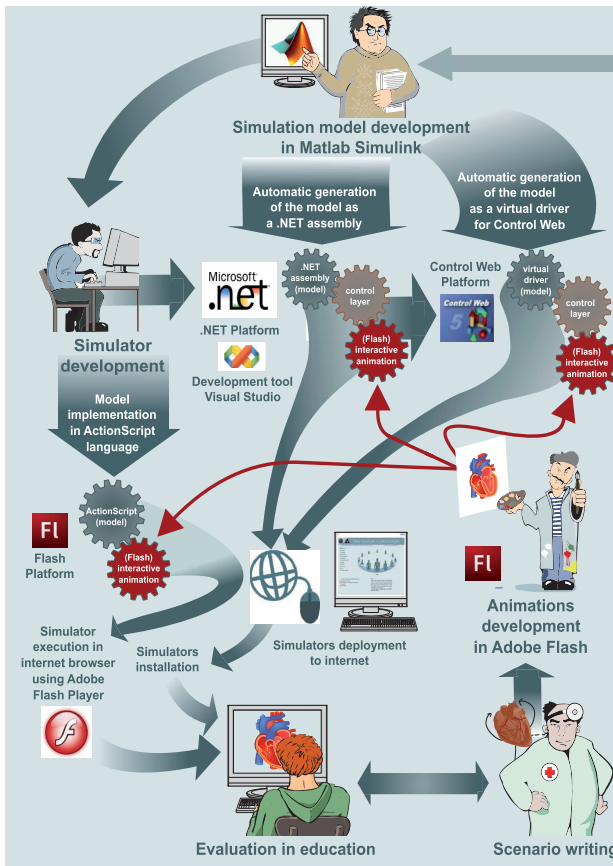


Figure 14: The original solution of creative interconnection of tools and applications, used for the creation of simulators and tutorial programmes using simulation games. The base of an e-learning program is a high-quality script, created by an experienced pedagogue. The creation of animated pictures is done by artists who create interactive animations in Adobe Flash. The core of simulators is the simulation model, created with special development tools, designed for the creation of simulation models. For a long time, we have been using Matlab/Simulink made by Mathworks for the development of models. The simulator development process is a demanding programming work. To make this task easier, we have developed special programmes that simplify the automatic transfer process of simulation models created in Matlab/Simulink over to ControlWeb or over to the Microsoft .NET environment.

road ahead from the identified model to the educational or tutorial simulator. It is a very demanding development work, which requires the combination of ideas and experiences of teachers who create the script of the tutorial application, the creativity of art designers who create the multimedia components interconnected with the simulation model in the background, as

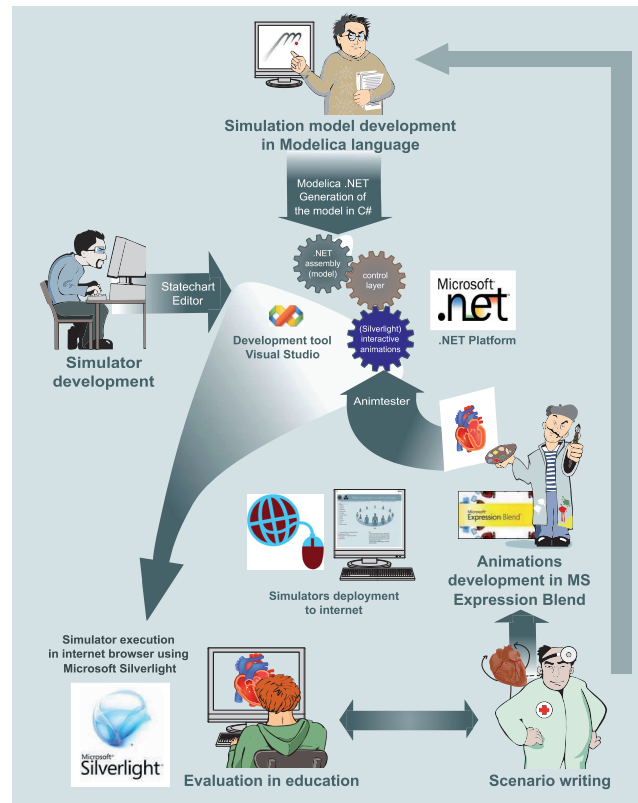


Figure 15: Our new solution of creative interconnection of tools and applications, used for the creation of simulators and tutorial programmes using simulation games. The base of an e-learning program is still a high-quality script, created by an experienced pedagogue. The creation of animated figures is done by artists who create interactive animations in Expression Blend. To create and test animations that will be controlled by the simulation model, art designers use the Animtester software tool, developed by us. The core of simulators is the simulation model, created in the Modelica simulation language environment. Within the project Open Modelica Source Consortium, we are in the process of creating a tool which is able to generate the source code from Modelica to C# language. This enables us to generate a component from .NET used in the final application on the Silverlight platform, which enables to distribute the simulator as a web application, running in the internet browser (even on computers with various operating systems).

well as the efforts of programmers who finally “sew up” the final masterpiece into its final shape. We have used a special web simulator creation technology for creation of educational simulators [20].

To automate the model debugging transfer from the simulation development environment (previously using Simulink and nowadays using Modelica) into

the development environment where the development application is programmed, specialized software tools (developed by us) are used. We have been creating tutorial simulators in Microsoft .NET and Adobe Flash environments (Figure 14). Recently, we began using the Microsoft Silverlight platform (Figure 15), which enables distribution of simulators over the internet and may be executed directly into the internet browser environment (even on computers running various operating systems).

7 Conclusions

Nowadays, the old Comenius motto – “schola ludus,” or “playful school” [6], has found a modern use in interactive educational programs that use simulation games. Connection of the multimedia environment, serving as an audio-visual user interface, with simulation models, gives the studied problem a much more tangible feeling. A simulation game offers the possibility to test, without any risk, the simulated object’s behavior. The behavior of individual physiological subsystems can be appreciated in a simulation game, both under normal conditions and in the presence of a disorder.

Complex integrative simulators of human physiology can be of large importance when teaching pathophysiology or studying pathogenesis of varied medical conditions and syndromes using virtual patients. Such simulators include models of not only individual physiological subsystems but also of their mutual connection into more complex units. Modelica is a very convenient developing tool for design of those complex hierarchical models.

References

- [1] Abram, S.R., Hodnett, B.L., Summers, R.L., Coleman, T.G., Hester R.L., Quantitative Circulatory Physiology: An Integrative Mathematical Model of Human Physiology for medical education. *Advanced Physiology Education*, 31 (2), pp.202 - 210, 2007.
- [2] Amosov, N.M, Palec, B.L., Agapov, G.T., Ermakova, I.I., Ljabach, E.G., Packina, S.A., Soloviev, V.P. *Theoretical research of physiological systems (in Russian)*. Naukova Dumaka, Kiev, 1977.
- [3] Bassingthwaite J. B., Strategies for the Physiome Project“, *Annals of Biomedical Engineering* 28, pp. 1043-1058, 2000.
- [4] Brugård, J., Hedberg, D., Cascante, M., Geder-sund, G., Gómez-Garrido, A., Maier, D., Nyman, E., Selivanov, V., Stralfors, P., Creating a Bridge between Modelica and the Systems Biology Community, *Proceedings 7th Modelica Conference, Como, Italy, Sep. 20-22, 2009.*, pp. 473-479, The Modelica Association., Como, 2009. Available <http://www.ep.liu.se/ecp/043/052/ecp09430016.pdf>.
- [5] Cellier, F. E., Nebot, A., Object-oriented modeling in the service of medicine, *Proceedings of the 6th Asia Conference, Beijing, China 2006*. 1, pp 33-40. International Academic Publishers, Beijing, 2006.
- [6] Comenius, J. A., *Schola ludus seu Encyclopaedia Viva.*, Sarospartak, 1656.
- [7] Fencl, J., Jabor, A., Kazda, A., Figge, J., Diagnosis of metabolic acid-base disturbances in critically ill patients, *Am. J. Respir. Crit. Care.*, 162, pp. 2246-2251, 2000.
- [8] Guyton A. C., Coleman T. A., Granger H. J., Circulation: Overall Regulation, *Ann. Rev. Physiol.*, 41, 13-41, 1972.
- [9] Haas, O. C., Burnham, K. J., Systems Modeling and Control Applied to Medicine, in O. C. Haas, K. J. Burnham, *Intelligent and Adaptive Systems in Medicine*, pp. 17-52, CRC Press, Boca Raton FL, 2008.
- [10] Hester R. L., Coleman T., Summers, R., A multilevel open source model of human physiology.“*The FASEB Journal*, 22, p. 756, 2008
- [11] Hester, R.L, Ilescu, R., Summers R. L , Coleman T. G., Systems biology and integrative physiological modeling, *Journal of Physiology*, Published online before print December 6, 2010, doi: 10.1113/jphysiol.2010.201558, Available: <http://jp.physoc.org/content/early/2010/12/01/jphysiol.2010.201558.full.pdf+html>.
- [12] Hunter P.J., Robins, P., Noble D., The IUPS Physiome Project, *Pflugers Archive-European Journal of Physiology*, 445, pp. 1–9, 2002.
- [13] Ikeda N., Marumo F., Shirsataka M. A., Model of overall regulation of body fluids, *Ann. Biomed. Eng.* 7, pp. 135-166, 1979.
- [14] Kofránek, J., Modelling of blood acid base balance, *Ph.D, Thesis*, Charles University, Faculty of General Medicine, Prague, 1980.
- [15] Kofránek, J., Complex model of acid-base balance (in Czech)., in M. Zeithamlová (Editor), *MEDSOFT 2009*, Praha: Agentura Action M., pp. 23-60. English translation of the paper is

- available at <http://www.physiome.cz/references/medsoft2009acidbase.pdf>, model is available at <http://www.physiome.cz/acidbase>.
- [16] Kofránek, J., Andrlík, M., Kripner, T., Mašek, J., Simulation chips for GOLEM – multimedia simulator of physiological functions, in J. G. Anderson, M. Kapzer (Editor), *Simulation in the Health and Medical Sciences 2002*. pp. 159-163, Society for Computer Simulation International, Simulation Councils, San Diego, 2002. Available: <http://www.physiome.cz/references/simchips2002.pdf>.
- [17] Kofránek J. Anh Vu L. D., Snášelová H., Kerekeš R. and Velan T., GOLEM – Multimedia simulator for medical education, in *MEDINFO 2001, Proceedings of the 10th World Congress on Medical Informatics. (London, UK, 2001)*, Patel, L., Rogers, R., Haux R. Eds., pp. 1042-1046, IOS Press, London, Available: <http://www.physiome.cz/references/medinfo2001.pdf>.
- [18] Kofránek, J., Mateják, M., Matoušek, S., Privitzer, P., Tribula, M., & Vacek, O., School as a (multimedia simulation) play: use of multimedia applications in teaching of pathological physiology. *MEFANET 2008. CD ROM Proceedings*, (ISBN 978-80-7392-065-4), kofranek.pdf: pp. 1-26, Masarykova Univerzita, Brno, 2008. Available: <http://www.physiome.cz/references/mefanet2008.pdf>.
- [19] Kofránek, J., Mateják, M. Privitzer, P., Tribula, M., Causal or acausal modeling: labour for humans or labour for machines, in V C. Moler, A. Procházka, R. Bartko, M. Folin, J. Houška, P. Byron (Editor), *Technical Computing Prague 2008, 16th Annual Conference Proceedings, CD ROM*, 058_kofranek.pdf: pp. 1-16. Humusoft s.r.o., Praha, 2008, Available: <http://www.physiome.cz/references/tcp2008.pdf>.
- [20] Kofránek, J. Mateják, M., Privitzer, P.: Web simulator creation technology. *MEFANET report*, 3, pp. 32-97. Available: <http://www.physiome.cz/references/mefanetreport3.pdf>.
- [21] Kofránek, J., Matoušek, S., Andrlík, M., Border flux ballance approach towards modelling acid-base chemistry and blood gases transport, in V B. Zupanic, S. Karba, S. Blažič (Editor), *Proceedings of the 6th EUROSIM Congress on Modeling and Simulation, Full Papers (CD)* (TU-1-P7-4, pp. 1-9), University of Ljubljana, Ljubljana 2007. Available: <http://www.physiome.cz/references/ljubljana2007.pdf>.
- [22] Kofránek, J., Matoušek, S., Rusz, J., Privitzer, P., Mateják, M., Tribula, M., The Atlas of physiology and pathophysiology: web-based multimedia enabled interactive simulations, *Computer Methods and Programs in Biomedicine*, Article in Press, doi:10.1016/j.cmpb.2010.12.007, 11 pp., 2011, Available: <http://www.physiome.cz/references/CMPB2011.pdf>.
- [23] Kofránek, J, Rusz, J., Restoration of Guyton diagram for regulation of the circulation as a basis for quantitative physiological model development *Physiological Research*, 59, pp 897-908, 2010, Available: http://www.biomed.cas.cz/physiolres/pdf/59/59_897.pdf.
- [24] Logan, J. D., Wolesensky, J. D., *Mathematical methods in biology*. John Wiley & Sons, Inc., Hoboken, NJ, 2009.
- [25] McLeod, J., PHYSBE: A ophysiological simulation benchmark experiment, *Simulation*, 15: pp. 324-329, 1966.
- [26] McLeod, J., PHYSBE...a year later, *Simulation*, 10, pp. 37-45, 1967.
- [27] McLeod, J., Toward uniform documentation- PHYSBE and CSMP, *Simulation*, 14, pp. 215-220, 1970.
- [28] Oomnes, C., Breklemans, M., Baaijens, F., *Bio-mechanics: concepts and computation*. Cambridge University Press, Cambridge, 2009.
- [29] Sirker, A. A., Rhodes, A., Grounds, R. M., Acid-base physiology: the 'traditional' and 'modern' approaches, *Anesthesia*, 57, pp. 348-356, 2001.
- [30] Stewart, P. A., Modern quantitative acid-base chemistry. *Can. J. Physiol. Pharmacol.*, 61, 1444-1461, 1983.
- [31] Thomas, R. S., Baconnier, P., Fontecave, J., Francoise, J., Guillaud, F., Hannaert, P., Hermandéz, A., Le Rolle, V., Mazière, P, Tahi, F., White R. J., SAPHIR: a physiome core model of body fluid homeostasis and blood pressure regulation, *Philosophical Transactions of the Royal Society*, 366, pp. 3175-3197, 2008.
- [32] Wallish, P., Lusignan, M., Benayoun, M., Baker, T. I., Dickey, A. S., Hatsopoulos, N. G., *MAT-LAB for Neuroscientists: An Introduction to Scientific Computing in MATLAB*. Academic Press, Burlington, MA, 2008.

Acknowledgement

Work on the development of medical simulators has been supported by the grants MPO FR-TI3/869, MSM 0021620806 and by Creative Connections s.r.o.

Modeling and Simulation of Heavy Truck with MWorks

Ying Sun, Wei Chen, Yunqing Zhang, Liping Chen
CAD Center, Huazhong University of Science and Technology, China
zhangyq@hust.edu.cn

Abstract

This paper shows the modeling and simulation of fuel economy and vehicle dynamics of a heavy truck with Modelica. The work was carried out on MWorks, which is developed by Huazhong University of Science & Technology. Comparisons between measured data and simulation results validate the correctness of the model, and demonstrate that Modelica can be used in the modeling and simulation of vehicle performance, and also shows the effectiveness MWorks software.

Keywords: Fuel Economy; Vehicle Dynamics; Modeling; Simulation; Modelica; MWorks

1 Introduction

The modeling and simulation of vehicle performance such as fuel economy, acceleration capability, handling, etc., are always carried out by some different software. For example, the fuel my, acceleration capability and gradeability are always analysis by AVL-Cruise, ADVISOR or some other software [1]. Vehicle dynamics is the science that studies the kinetics and dynamics of wheeled land vehicles. The modeling and simulation of vehicle dynamics are always carried out by some multibody system software such as MSC.ADAMS, Simpack, Carsim, etc [2]. However, these simulation software are only strong in one domain and are not capable to model components from other domains in a reasonable way. This is a major disadvantage since technical systems are becoming more and more heterogeneous with components from many engineering domains [3].

Modelica has been used widely in various domains, and shows the ability in the modeling of complex physical systems. Modelica is a modern language used to model physical systems. The language is object-oriented, non-causal and the models are mathematically described by differential algebraic equations. The characteristic of modelica language make it very suited to define model libraries with reusable

components, model complex applications involving parts from several application domains, and many more useful facilities [4].

This paper shows the modeling and simulation a heavy truck with MWorks based on unified modeling language Modelica. Comparisons between measured data and simulation results validate the correctness of the model, and demonstrate that Modelica can be used in the modeling and simulation of vehicle performance, and also shows the effectiveness MWorks software.

MWorks is under developed by Huazhong University of Science and Technology. It is a general modeling and simulation platform for complex engineering systems which supports visual modeling, automatically translating and solving, as well as convenient postprocessing. The current version is based on Modelica 2.2 and implements almost all the syntax and semantics of Modelica. More details about MWorks can be referred to [5].

2 Power Transmission Library

The power transmission library is designed to simulation of the automotive driving performance, such as fuel economy, acceleration capability, gradeability, etc. The library includes the main power transmission components of a heavy truck, such as engine, clutch, gears, final drive, wheel, vehicle, brake, driver, etc., and can be seen in Fig. 1.

The components of the power transmission library are shortly described below:

➤ Engine

The component engine contains a model for a combustion engine. As the characteristic curves for the full load, the fuel consumption and others can be freely defined by the user. It is possible to define a gasoline engine as well as a diesel engine. The engine will be modeled by a structure of characteristic curves and maps.

➤ Clutch

The clutch contains the model of a friction clutch as used in cars with manual gear boxes. It is controlled by the driver via the clutch pedal position.

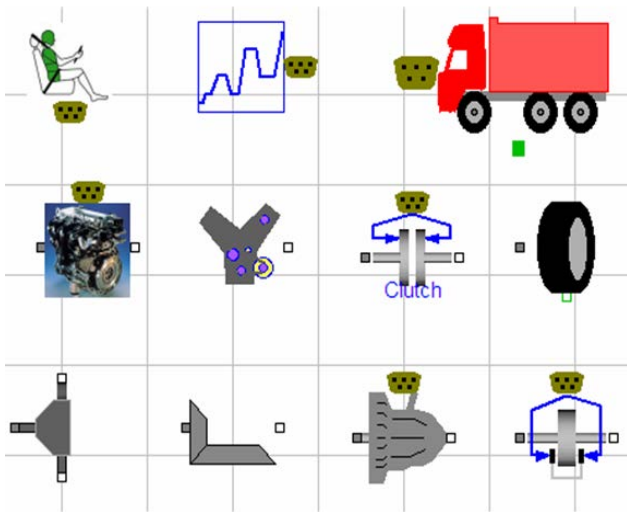


Fig.1 Components of Power Transmission Library

➤ Gears

The gear package includes gears in the vehicle, such as gearbox, differential gear, final drive, etc. The gears are modeled as gear ratios, and the inertial of the gear and moment loss are considered.

➤ Wheel

The wheels and tires link the vehicle to the road. This component allows to consider many influencing variables and their effect on the rolling state.

➤ Vehicle

The vehicle is one of the main objects in a model. This component contains general data of the vehicle, such as nominal dimensions and weights. The library presents only dynamic models for the longitudinal motion of the vehicle. So a sliding mass may represent as vehicle body.

➤ Driver

The driver includes gearbox control, clutch control brake control, throttle control and so on.

➤ DriveCycle

The drive cycle model includes some drive cycles, such as EDC, UDDC, NYCC driving cycles, etc.. Some other cycles can be added if desired.

➤ Brake

This is described by braking data and dimensions. By the implementation of a specific braking factor

it is possible to model disc brakes as well as different forms of drum brakes.

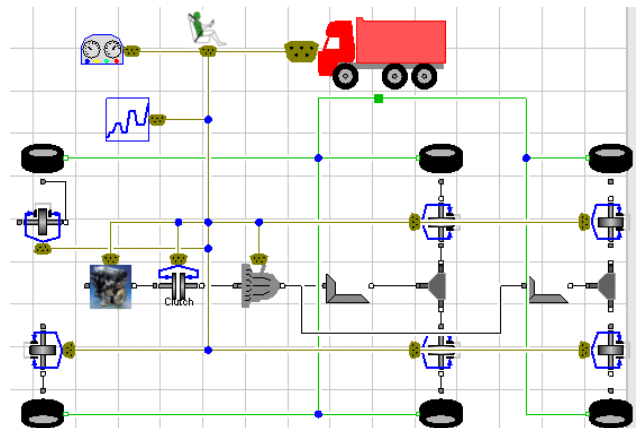


Fig.2 Model of the Heavy Truck

The user can build a heavy truck model very easily, seen in Fig. 2. The user drags the components together and connects lines in the graphic user interface. The vehicle was modeled by a forward-facing approach include the driver model, which controls the throttle, brake, clutch, gearbox to make the vehicle speed follow a given driving cycle.

3 Vehicle Dynamic Library

The vehicle dynamic library is designed to simulation of the dynamic performance such as handling, K&C of the heavy truck. The library was built based on standard library. The wheel model and bushing model used in this paper are from VehicleDynamics Library 0.8 which is designed by Modelon [6] and made some modification. As can be seen in Fig.3, the subsystem is built by basic components, and the heavy truck was assembled by the subsystems.

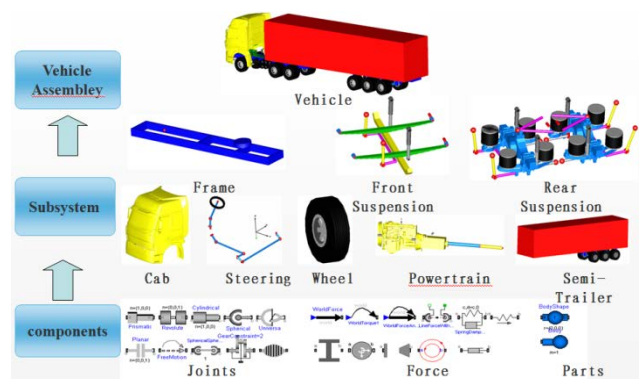


Fig.3 Vehicle Dynamic Library

The standard library MultiBody is a free Modelica package providing 3-dimensional mechanical components such as joints, functions, forces, parts, sen-

sors, etc., to model mechanical systems. However, it's not enough to carry out vehicle dynamic modeling and simulation. Many important components such as nonlinear spring, nonlinear damper, bushing, beam, airspring, tires, etc., don't exist in the standard modelica library. So we must extend the mutibody library to carry out vehicle dynamic modeling and simulation.

The heavy truck with ten-wheel three-axle was designed and manufactured by China National Heavy Duty Truck Group Corp., Ltd. The truck has conventional steering system, leafspring front suspension and airspring rearsuspension, and will be described in the following.

3.1 Leaf Spring Modeling

Leaf springs are commonly used in the suspension systems of heavy truck. Accurate modeling of the leaf springs is necessary in evaluating ride comfort, braking performance, vibration characteristics, and stability [7]. There several ways to model the leaf spring. In this paper, simple beam theories are used to model the dynamics of the leaf spring. The beam elements have the constant cross section. A total of 17 elements were used to create the single leaf spring model, seen in Fig.4. The shackle and the leaf seat are modeled as rigid parts. The pin joints are represented by revolute joints with one degree of freedom along the global y-axis. The forces and moments are applied at the axle seat. To decrease the amount of the equations and increase the calculation efficiency, the frictions between the leaves and the contact force are neglected.

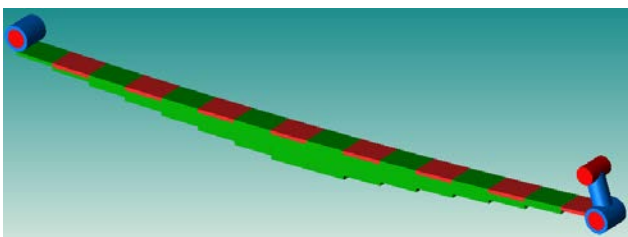


Fig.4 Leaf Spring Model

3.2 Front Suspension Modeling

The front suspension includes leaf spring, antiroll bar, axle and the shock absorber. According to the topology, the front suspension was build very easily, seen in Fig.5. The port F is connected to frame, the port LS is connected to the left part of the steering system, and the port RS is connected to the right part of the steering system.

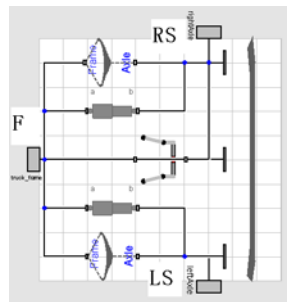


Fig. 5 Front Suspension

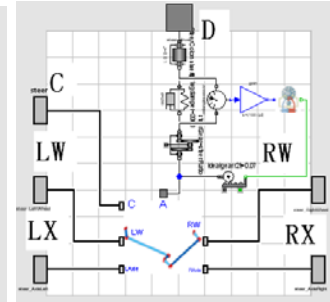


Fig. 6 Steering System

3.3 Steering Modeling

The steering system includes a steering column and steering trap. In this paper, the steering column is modeled as a inertial for simplification, and the steering trap is modeled according to the topology. The steering column and the steering trap is connected by ball screw, which is modeled by IdealGearR2T in the Modelica library. The hydraulic power steering is essential to the heavy truck to assist the driver, and also modeled by hydraulic components. As can be seen in Fig.6, the port D is connected to the driver model, the port LX and RX is connected to the left and right axle, and the port LW and RW is connected to the left and right wheel.

3.4 Rear Suspension Modeling

The rear suspension includes v-rods, push-rods, air springs, antiroll and shock absorber. According to the topology, the rear suspension was build, seen in Fig.7. The port F is connected to frame, the port LW is connected to the left wheel, and the port RW is connected to the right wheel.

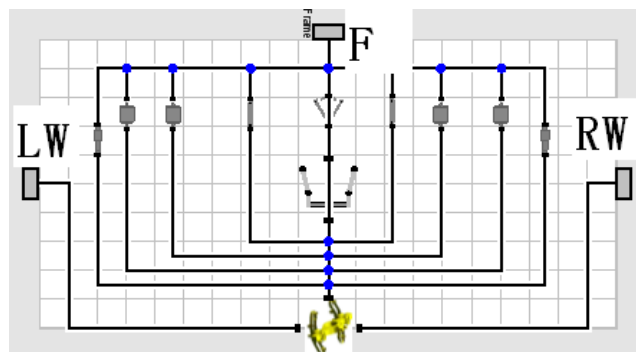


Fig. 7 Rear Suspension System

3.5 Vehicle Assembly

Based on the subsystem model, the user can build subsystem assembly and vehicle assembly very easily, seen in Fig.8 and Fig.9. The user drags the components together and connects lines in the graphic user interface. The Model includes the front suspen-

sion, steering system, the middle and rear bridle, the frame, the cab, powertrain, the driver, wheels and the semitrailer. Based on the model, the K&C and the handling can be carried out to analysis the performance of the heavy truck.

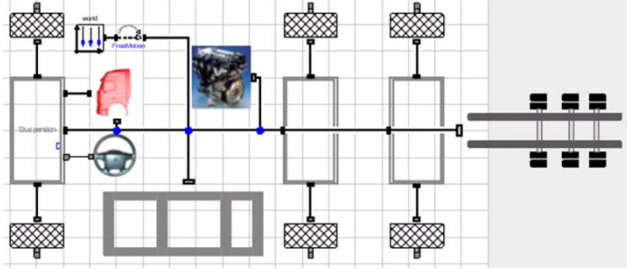


Fig. 8 Heavy Truck Assembly

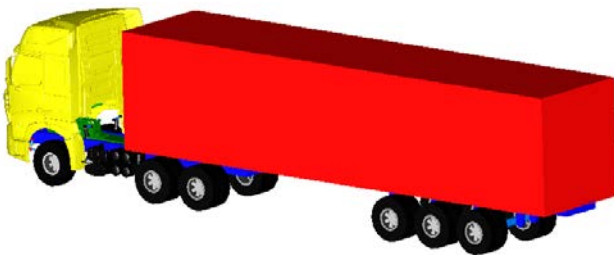


Fig. 9 3D View of Heavy Truck model

4 Simulation and Results

To analysis the performance the heavy truck, the simulation about the fuel economy, the K&C of the subsystem and the handling performance were carried out in the following.

4.1 Fuel Economy Simulation

Fuel consumption is the amount of fuel used per unit distance. Lower values mean better fuel consumption. The key parameters of the vehicle can be seen in Table 1. The simulation was carried out to calculated the fuel consumption with constant speed in 15th and 16th gear. The simulation results can be seen in Fig.10 and Fig.11.

Table 1 Key parameters of the vehicle

Components	Key Parameters
Engine	Maximum power: 1802kW@1500rpm Maximum torque: 309Nm@2000r/min
Final Drive	3.93
Transmission	15.59/13.12/10.89/9.17/7.48/6.30/5.20/ 4.38/3.56/3.00/2.49/2.10/1.71/1.44/1.19 /1.00
Vehicle Mass	49000Kg
Wheel Radius	0.548
Rolling Resistance	0.0091
Frontal Area	8.5m ²

Coefficient of Aerynamic Drag	0.585
-------------------------------	-------

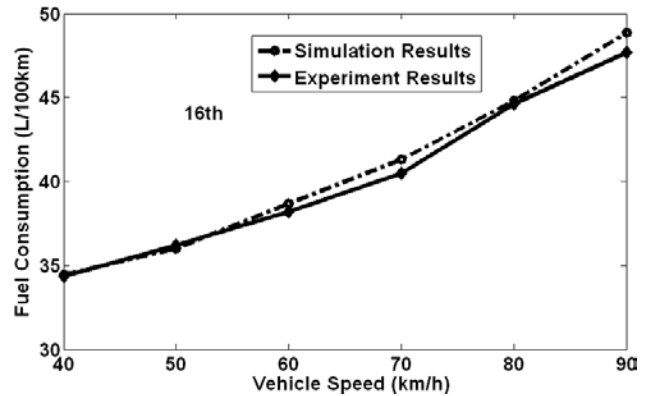


Fig. 10 Fuel Consumption with Constant Speed in 16th Gear

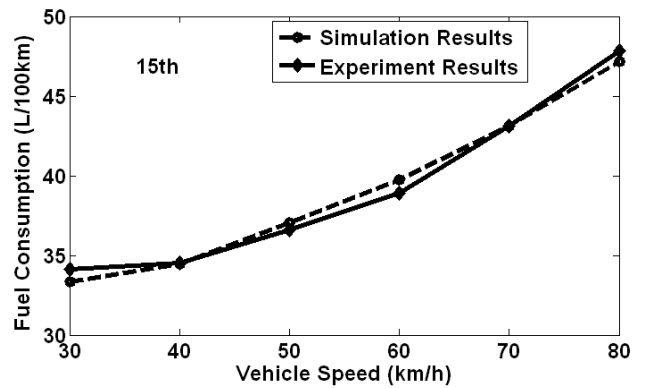


Fig.11 Fuel Consumption with Constant Speed in 15th gear

The dashed line is the experiment data, and the solid line is the simulation data. The figures showed that the simulation results were very close to the experiment data.

4.2 Vehicle Dynamic Simulation

With the vehicle dynamic library, the vehicle be built very easily, and some open loop test can be simulated, such as step steer, impulse steer, brake, etc.

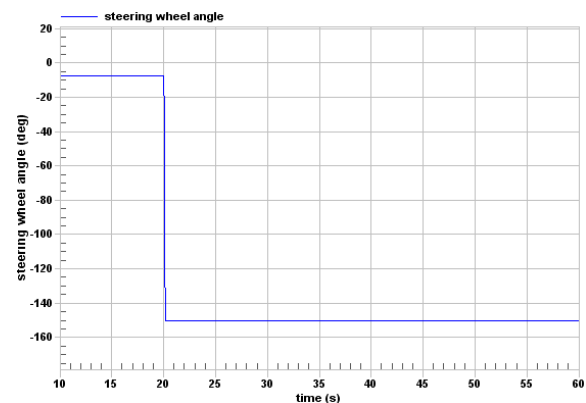


Fig.12 Steering Wheel Angle

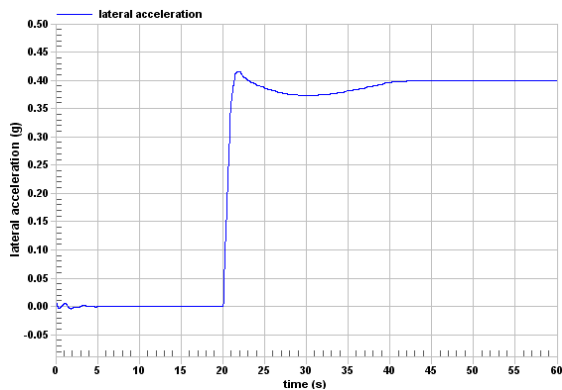


Fig.13 lateral acceleration

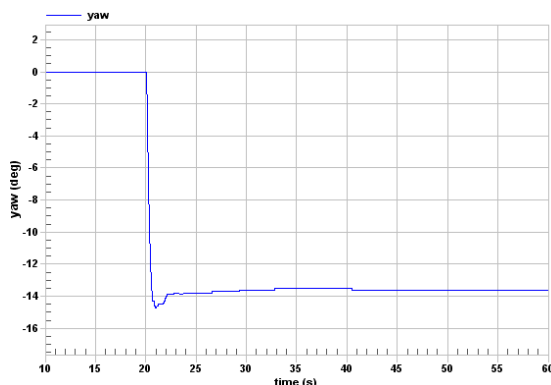


Fig.14 Yaw speed

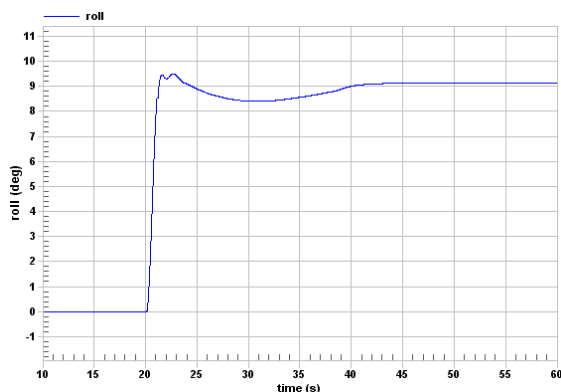


Fig.15 Roll Angle

The step steering simulation was carried out. The steering angle and vehicle speed were considered as input, seen in Fig.12. Fig.13-Fig. 15 showed the dynamic performance of the vehicle.

5 Conclusions

This paper shows the modeling and simulation of fuel economy and vehicle dynamics of a heavy truck with Modelica. The work was carried out on MWorks. Comparisons between measured data and simulation results validate the correctness of the model, and demonstrate that Modelica can be used in the modeling and simulation of ve-

hicle performance, and also shows the effectiveness MWorks software.

Acknowledgement

This work was supported by the National High-Tech R&D Program, China (No. 2009AA044501).

References

- [1] Wang J., etc. Forward simulation model precision study for hybrid electric vehicle. *Mechatronics and Automation*, 2009 , pp. 2457 – 2461.
- [2] Blundell M., Harty D. *The Multibody Systems Approach to Vehicle Dynamics*. SAE, Warrendale, 2004.
- [3] Hilding Elmqvist. *An Introduction to the Physical Modeling Language Modelica*. Proceedings of the 9th European Simulation Symposium, ESS'97, Oct 19-23, 1997, Passau, Germany.
- [4] Modelica Association. <http://www.modelica.org>.
- [5] FAN-LI Zhou, LI-PING Chen, etc. MWorks: a Modern IDE for Modeling and Simulation of Multidomain Physical Systems Based on Modelica. Proceedings of the 5th International Modelica Conference, 2006.
- [6] J. Andreasson and M. Gafvert. *The VehicleDynamics Library-Overview and Applications*. Proceedings of the 5th International Modelica Conference, 2006.
- [7] Hiroyuki Sugiyama, etc. Development of nonlinear elastic leaf spring model for multibody vehicle systems. *Computer Methods in Applied Mechanics and Engineering*, 2006, pp. 6925-6941.

Object-Oriented Electrical Grid and Photovoltaic system modelling in Modelica

Bart Verbruggen¹ Juan Van Roy¹ Roel De Coninck^{2,4} Ruben Baetens³
Lieve Helsen² Johan Driesen¹

¹ Department of Electrical Engineering, K.U.Leuven, B-3000 Leuven, Belgium

² Department of Mechanical Engineering, K.U.Leuven, B-3000 Leuven, Belgium

³ Department of Civil Engineering, K.U.Leuven, B-3000 Leuven, Belgium

⁴ 3E, B-1000 Brussels, Belgium

Abstract

In this paper an object-oriented model for an electrical grid based on the IEEE 34 node test feeder and the integration of this grid with photovoltaic systems on residential level, i.e. 230 V, is presented. The goal is to demonstrate the flexibility of using Modelica to simulate a grid model and to show how this model can be used to simulate the influence of a household power demand and photovoltaic generation on this electrical grid. The idea is to clarify the voltage problem that might occur when implementing a large amount of photovoltaic (PV) systems into a residential grid.

Keywords: Electrical grid; Photovoltaic system; Residential building

1 Introduction

The models that are presented in this paper are a direct result of work that has been accomplished within the K.U.Leuven Energy Institute (EI) project *Optimized Energy Networks for Buildings*. This project focuses on the energy flows within a building and a group of buildings. These energy flows include thermal and electrical energy transfers combined with internal energy conversions. The developed models focus on the electrical aspect, the aspects of the building itself and the thermal installations in the building. These different aspects are modeled by different departments of the Faculty of Engineering at the K.U.Leuven. An object-oriented approach to modelling the building(s) with its interactions is advisable. This way, each model can be developed and tested in their respective domains, before interconnecting them. This paper only treats the electrical part.

The example, that is presented in this paper, is

developed to illustrate how the integration of the grid and photovoltaic system (PV) models can be accomplished. The choice for an object-oriented approach together with the multi-disciplinary nature of the project has led to the use of Modelica.

First, a photovoltaic (PV) system has been implemented based on the five parameter model, as a part of the electrical models being developed [1]. The parameters needed for this model can generally be obtained from data gathered from the manufacturer's specifications of the solar panels.

A second electrical model has been developed for an electrical distribution grid at low voltage (i.e. 230 V). This electrical distribution grid is based on the topology of the IEEE 34-bus model developed for a medium-voltage industrial 24.9 kV distribution grid [2]. The parameters of this distribution grid have been downscaled to represent a residential radial low-voltage electrical distribution network. The object-oriented model has been developed in such a way that the distribution grid topology and its impedances are characterized by a Modelica Record, ensuring maximum flexibility and scalability in the use of the model. As such, this model can be used to describe intra-building grids, which are radial as well. DC-networks follow the same methodology.

2 Photovoltaic system model

2.1 Introduction to the five parameter model

The five parameter model, which is temperature dependent, is based on the single diode equivalent circuit of a PV panel [3]. The five parameters used are:

- the light current I_{ph}

- the diode reverse saturation current I_0
- a shunt resistance R_{sh}
- a series resistance R_s
- the thermal voltage V_t

These parameters are indicated in the equivalent circuit presented in Figure 1.

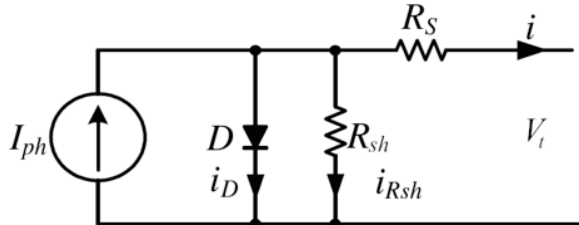


Figure 1: Five parameter model of a PV panel [3]

2.2 Calculation of the five parameters

The five parameters in the model (I_{ph} , I_0 , R_{sh} , R_s and V_t) can be calculated based on characteristics that are provided by the solar panel manufacturer. The necessary specifications to calculate the five parameters are the current I_{mpp} and voltage V_{mpp} at maximum power point (mpp) under standard testing conditions (STC), the short circuit current I_{sc} and open circuit voltage V_{oc} under the same standard testing conditions and the temperature coefficients k_i and k_v of respectively the short circuit current and open circuit voltage. Based on these specifications, Sera et al. [4] gives the calculation method for the five parameters.

The general current-voltage ($i - v$) equation for the single diode equivalent circuit is given in Eq. (1). In this equation V_t is the junction thermal voltage and n_s the number of cells in the panel connected in series. The voltage V_{mpp} and current I_{mpp} at maximum power point should satisfy this equation and according to Eq. (2), the derivative of the power with respect to the voltage should be zero at this point. Eq. (3) states that the derivative of the current with respect to the voltage at short circuit current should be the negative of the shunt conductance ($1/R_{sh}$). These equations lead to the calculation of the parameters R_s , R_{sh} and V_t .

$$i = I_{ph} - I_0 \cdot \left(e^{\frac{v+i \cdot R_s}{n_s \cdot V_t}} - 1 \right) - \frac{v+i \cdot R_s}{R_{sh}} \quad (1)$$

$$\left. \frac{dP}{dV} \right|_{\substack{V=V_{mpp} \\ I=I_{mpp}}} = 0 \quad (2)$$

$$\left. \frac{dI}{dV} \right|_{I=I_{sc}} = -\frac{1}{R_{sh}} \quad (3)$$

The reverse saturation current I_0 and light current I_{ph} at STC can be found based on Eq. (1) for the short circuit (Eq. (4)) and open circuit condition (Eq. (5)).

$$I_{sc} = I_{ph} - I_0 \cdot e^{\frac{I_{sc} \cdot R_s}{n_s \cdot V_t}} - \frac{I_{sc} \cdot R_s}{R_{sh}} \quad (4)$$

$$I_{oc} = 0 = I_{ph} - I_0 \cdot e^{\frac{V_{oc}}{n_s \cdot V_t}} - \frac{V_{oc}}{R_{sh}} \quad (5)$$

2.3 Five parameter PV model in Modelica

The five parameter model is implemented in a Modelica model to calculate the power output of the photovoltaic panels under operational conditions. The current and voltage at maximum power point can be found by solving Eqns. (1) and (2) for the non-reference conditions. The parameters for these conditions are calculated in the next paragraphs.

The PV parameters are adjusted to take into account the position of the sun, the direct and indirect radiation and the ambient temperature. The cell temperature has been adjusted to be the ambient temperature plus the losses of the panel.

The tilt angle and orientation of the PV panels are parameters of the PV model. Together with the sun's position, the incidence angle of the direct beam radiation can be calculated which allows to obtain the amount of radiation that gets reflected by and passes through the PV panel cover. This is done using incidence angle modifiers that are derived from De Soto et al. [3]. The incidence angle modifier $K_{\tau\alpha}(\theta)$ can be found from the transmittance τ of the cover system with Eq. (8), which is approximated in Eq. (7). The angle of refraction, θ_r , is determined in Eq. (6) by Snell's law, with θ the incidence angle and n the effective index of refraction of the cell cover. In Eq. (7), K is the glazing extinction coefficient and L is the glazing thickness. In the model K and L can be adjusted. By default, K is assumed to be 4 m^{-1} and L is assumed to be 2 mm .

$$\theta_r = \arcsin(n \cdot \sin\theta) \quad (6)$$

$$\tau(\theta) = e^{-\frac{K \cdot L}{\cos\theta_r}} \cdot \left[1 - \frac{1}{2} \cdot \left(\frac{\sin^2(\theta_r - \theta)}{\sin^2(\theta_r + \theta)} + \frac{\tan^2(\theta_r - \theta)}{\tan^2(\theta_r + \theta)} \right) \right] \quad (7)$$

$$K_{\tau\alpha}(\theta) = \frac{\tau(\theta)}{\tau(0)} \quad (8)$$

The incidence angle modifiers and the direct and diffuse radiation, which are inputs to the model, allow

together with the reflected radiation to calculate the absorbed solar radiation S in Eq. (9). In this equation G_b is the direct, G_d the diffuse and G the total radiation. The slope of the PV panel is characterized by β .

$$\frac{S}{S_{ref}} = \frac{G_b}{G_{ref}} \cdot K_{\tau\alpha,b} + \frac{G_d}{G_{ref}} \cdot K_{\tau\alpha,d} \cdot \frac{1 + \cos\beta}{2} + \frac{G}{G_{ref}} \cdot \rho \cdot K_{\tau\alpha,g} \cdot \frac{1 - \cos\beta}{2} \quad (9)$$

The light current I_{ph} , reverse saturation current I_0 and thermal voltage V_t at non-reference conditions can be calculated when the temperature, open circuit voltage and short circuit current are known [4]. The open circuit voltage V_{oc} can be calculated using Eqns. (10) and (11). The short circuit current I_{sc} can be found using Eq. (12).

$$e^{\frac{V_{oc}(S)}{n_s \cdot V_t}} = \frac{I_{ph}(S) \cdot R_{sh} - V_{oc}(S)}{I_0 \cdot R_{sh}} \quad (10)$$

$$V_{oc}(T) = V_{oc} + k_v \cdot (T - T_{stc}) \quad (11)$$

$$I_{sc}(S, T) = I_{sc} \cdot \left(\frac{S}{S_{ref}} \right) \cdot \left(1 + \frac{k_i}{100} \cdot (T - T_{ref}) \right) \quad (12)$$

The reverse saturation current I_0 can be calculated with Eq. (13). The light current I_{ph} is found using Eq. (14).

$$I_0 = \left(I_{sc} - \frac{V_{oc} - I_{sc} \cdot R_s}{R_{sh}} \right) \cdot e^{-\frac{V_{oc}}{n_s \cdot V_t}} \quad (13)$$

$$I_{ph} = I_0 \cdot e^{\frac{V_{oc}}{n_s \cdot V_t}} + \frac{V_{oc}}{R_{sh}} \quad (14)$$

The ambient temperature and the direct and diffuse radiation are read from a meteorological data file and can be given as parameters to the model of the photovoltaic panel. This meteorological data are taken from Meteororm 6.1 and give minute values for solar radiation and ambient temperature [5]. The direction of the beam radiation to the PV panel and the position of the sun are calculated in another model based on the simulation time.

2.4 Conclusion

Since the model of the PV panel takes I_{ph} , I_0 , R_{sh} , R_s and V_t as parameters, it can be used to model a large variety of PV panels for which these parameters are known. For most panels they can be calculated from manufacturer catalogue data, for which the method is described in section 2.2. The PV panel model calculates the power, current and voltage of the panel

and can be used to put several panels in series or in parallel. This leads to an optimal flexibility for the use of this model.

3 Electrical grid model

The goal of this paper is to demonstrate a working grid model to show the effects of the power demand of a household and the power generation of a PV system on a distribution grid.

First, the grid topology is described in section 3.1. This section discusses the simulated grid, the representation of the grid in matrix form and how this matrix form allows to model any radial grid topology.

Traditionally, electrical grids are examined using a power flow analysis to determine the nodal currents, line currents and nodal voltages. The background on power flow analysis is given in section 3.2. Next, section 3.3 gives an overview of the object-oriented implementation of this problem in Modelica. The grid is implemented in Modelica making full use of the object-oriented modelling language for component-oriented modelling of complex systems. A simple grid with two households and two PV systems is considered to illustrate schematically the use of the model in Modelica.

In section 4 the full IEEE 34 node grid is used to analyze the grid for any voltage problems that may occur when there is a certain penetration rate of photovoltaic systems in the grid.

3.1 Grid topology

In this paper a grid model is used on a district level for residential connections. Figure 2 shows the topology of the modelled IEEE 34 node test feeder [2]. This type of distribution grid is typically a radial network with a rated voltage of 230 V (400 V wye, or star, connection). In a radial network, there is only a single-feeder transformer, thus the reduced reliability is the major disadvantage, due to the lack of a redundant supply. An interruption at one node means all nodes downstream are interrupted. The IEEE 34 node test feeder, used in this paper, is based on a distribution grid of Arizona (USA) with a rated voltage of 24.9 kV. The voltage is downscaled to 230 V (Europe). Therefore, also the line impedances are adapted [6].

Each node in this grid represents a residential connection with a certain power demand. Besides it

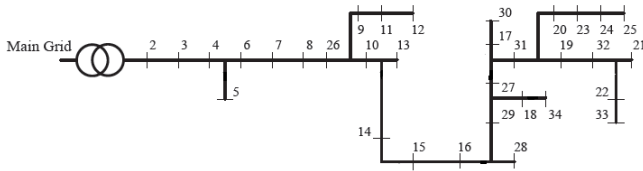


Figure 2: Grid topology IEEE 34 node test feeder [2]

is possible to have a random number of houses with a PV system.

The connections of the nodes of this grid can be represented by an incidence matrix (or connection matrix) \mathbf{T} . Each row in \mathbf{T} stands for a line between two nodes. Since there are 34 nodes in the grid, there are 33 lines. The number of columns equals the number of nodes. A line can then be represented by 1 and -1 at respectively the start and end node of the line. The other row elements are zero. To obtain a square matrix, an extra (first) row is introduced to represent the 'line' between the transformer and the first node, in which there is only an end node. Thus, in this example for the IEEE 34 node test feeder, the incidence matrix \mathbf{T} is a 34-by-34 matrix. This description with an incidence matrix \mathbf{T} basically allows any radial grid topology to be modelled.

Eq. (15) gives an example of the representation of an incidence matrix \mathbf{T} in which the nodes are all next to each other, which means branches are between consecutive nodes. In the example used in this paper (a radial grid as presented in Figure 2), the matrix is not as structured as in Eq. (15).

$$\mathbf{T} = \begin{pmatrix} -1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 1 & -1 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 1 & -1 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & -1 & 0 & 0 \\ 0 & 0 & 0 & \cdots & 1 & -1 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 1 & -1 \end{pmatrix} \quad (15)$$

The impedance matrix \mathbf{Z} represents the impedances of each line in the grid. The impedance of a line is defined as a combination of a real (resistance \mathbf{R}) and imaginary (reactance \mathbf{X}) part (Eq. (16)).

$$\mathbf{Z} = \mathbf{R} + j \cdot \mathbf{X} \quad (16)$$

3.2 Background of electrical grid modelling

The proposed grid can be used to perform a load flow analysis in order to characterize the impact of the load profile of each residential connection and

PV electricity generation on the grid. Traditionally, a power flow analysis is used to determine e.g. the voltage deviations at each node. This section gives the background of this method.

Several methods are available to determine the grid parameters in a power flow analysis. Generally, the *backward-forward sweep* is used. With this method it is possible to calculate the vector of nodal currents \mathbf{I}_{node} , line currents \mathbf{I}_{line} (between two nodes) and the nodal voltages \mathbf{U}_{node} for each time step. As such, the full flow of active power P and reactive power Q is known for each node. P and Q can be calculated from the apparent power S , using Eqns. (17)-(19), with \mathbf{I}^* the complex conjugate of \mathbf{I} .

$$S = P + j \cdot Q = U \cdot I^* \quad (17)$$

$$P = \Re(S) \quad (18)$$

$$Q = \Im(S) \quad (19)$$

The parameters (\mathbf{I}_{node} , \mathbf{I}_{line} , \mathbf{U}_{node}) are determined via iteration. In a first iteration a fixed voltage profile of 230 V for each node is assumed. For the next time steps this flat voltage profile for the first iteration step can be replaced by the voltage profile of the last simulated time step. When the time step is small enough, this will lead to a faster convergence of the iteration because this voltage profile will be closer to the actual one.

After this first iteration step, the backward-forward sweep will be used till convergence is reached. In the backward step, the nodal and line currents are calculated by Eqns. (20) and (21). Both currents are calculated based on parameters of the previous iteration step. In Eqns. (20) and (21), the nodal and line currents and the nodal voltages are complex. \mathbf{S}_{node} stands for the apparent power which consists of the active power (P) and reactive power (Q) (Eq. (17)). The incidence matrix \mathbf{T} is used to calculate the line currents. Note that the transpose of \mathbf{T} is taken in Eq. (21).

$$\mathbf{I}_{node} = f(\mathbf{S}_{node}, \mathbf{U}_{node}) = \left(\frac{\mathbf{S}_{node}}{\mathbf{U}_{node}} \right)^* \quad (20)$$

$$\mathbf{I}_{line} = (\mathbf{T}^T)^{-1} \cdot \mathbf{I}_{node} \quad (21)$$

In the forward step, the nodal voltage is calculated with the line currents calculated in Eq. (21). The nodal voltage given in Eq. (22) is then compared, just like all parameters, with the values from the previous iteration step. The iteration in Eqns. (20)-(22) starts again until convergence is reached (error: $\varepsilon \leq \varepsilon_{criterion} = 0.0001$).

$$\mathbf{U}_{node} = \mathbf{U}_{grid} - \mathbf{Z} \cdot \mathbf{I}_{line} \quad (22)$$

3.3 Electrical grid model in Modelica

The grid is implemented in Modelica making full use of the object-oriented modelling language for component-oriented modelling of complex systems. The implementation of the grid has been made very flexible by making use of the incidence matrix \mathbf{T} and the impedance matrix \mathbf{Z} . Only these two matrices have to be adapted when using another grid. This flexibility is thanks to the incidence matrix which contains the nodes that need to be connected for each line. Thus, the connection of the different nodes can be automated using this incidence matrix \mathbf{T} . Another advantage is the fact that the iteration for the calculation of the nodal and line currents and the nodal voltages is automatically carried out by Modelica, since Modelica is equation based.

The grid is set up as can be seen in the Modelica code presented in Code 1. This code builds the grid from different components and arrays of components. An array of nodes is used to facilitate the connection of different branches. These nodes are also useful to facilitate the connection of loads, electricity generation and households when the grid is used. An array of branches with impedances specified by the impedance matrix \mathbf{Z} is connected to the nodes in the for loop. This loop uses the incidence matrix \mathbf{T} to build up the network. Different grid layouts are being implemented as records containing the corresponding \mathbf{Z} and \mathbf{T} matrices. Components with a ground and a fixed complex voltage are used to complete the grid model.

To show how the usage of the node pins can greatly facilitate further use of the grid, the Modelica code in Code 2 is presented. As can be understood, the use of these nodes is not necessary, but it simplifies the implementation of larger grids. Nodes, household loads and PV systems are connected using a "pQtoVI" component (see further in this section), in which the grid side is connected to the "VI" pin, whereas households and PV systems are represented by a power flow. Households and PV systems are thus connected to the "PQ" pin.

The different components that are considered in the simulations in section 4 of this paper are further explained using a visual example of a simple grid with only two branches and households with PV systems, shown in Figure 3. This visual example uses the same components as the larger grid, with exception of the nodes, but is manually constructed. For larger grids, the Modelica code in Code 1 is used, supplemented with extra Modelica code, like the one in Code 2.

```

model Grid
  parameter ELECTA.Grids.MatVec grid;
  parameter Integer n=grid.n;
  parameter Integer T[n,n] = grid.T;
  parameter
    Modelica.SIunits.ComplexImpedance [n]
    Z = grid.Z;
  parameter SI.ComplexVoltage
    Vsource=230+0*j "Voltage";
  ELECTA.DisGrid.GridCom.CGround ground;
  ELECTA.DisGrid.GridCom.CVoltageSource
    source(Vsource=Vsource);
  ELECTA.DisGrid.GridCom.Branch branch [n]
    (R=Modelica.ComplexMath.real(Z),
     X=Modelica.ComplexMath.imag(Z));
  ELECTA.DisGrid.GridCom.CPosPin [n] node;

equation
  connect (ground.p, source.n);
  connect (branch[1].p, source.p);
  connect (branch[1].p, node[1]);
  for x in 1:n loop
    for y in 1:n loop
      if T[x,y]==1 then
        connect (branch[x].p, node[y]);
      elseif T[x,y]==-1 then
        connect (branch[x].n, node[y]);
      end if;
    end for;
  end for;
end Grid;

```

Code 1: Grid model

```

for y in 1:grid.n loop
  connect (pq[y].vi, grid.node[y]);
  connect (pq[y].pq, pv[y].pq);
  connect (pq[y].pq, h1[y].load_p);
end for

```

Code 2: Use of the grid model

A fixed source with a fixed complex voltage is used. The reference scenario is a voltage of 230 V. This means a DC power flow analysis will be performed. A DC power flow analysis is a static analysis which is a simplification of the transient AC power flow analysis. In an AC power flow analysis, a non-linear system describes the power flow through the lines (branches). In a DC power flow analysis a linear system is applicable. A DC power flow analysis is less accurate, but simulating all transients in an AC power flow analysis requires a much smaller simulation time step and thus a higher computation time. Furthermore, in a distribution grid (as given in

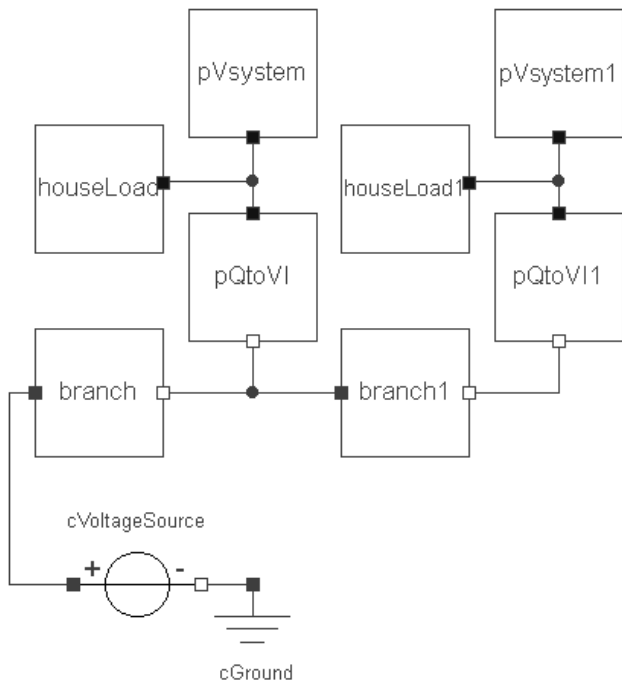


Figure 3: Visual example of a grid with two households with each a PV system

this paper) the resistance R of the lines is much higher than the reactance X , which justifies the use of a DC power flow analysis as a good approximation for an AC power flow analysis.

Each line between two nodes is represented by a branch which represents a line with a certain impedance Z (Eq. (16)).

The load profile of a household is represented in the component "houseLoad". The output of this component is an active power demand (see section 4.1) which will be fulfilled by the PV system and grid.

The PV system delivers an active power output which will be used to cover a part of the power demand of the household or will be put partly on the grid.

The difference between the power output of the PV system and the power demand of the household defines how much power is needed from the grid or will be injected in the grid. The component "pQtoVI" calculates with the fixed active power (to or from the household and PV system) the nodal voltage and nodal current (current to or from the household and PV system) at the node on the grid. Modelica takes care of the iteration to define the voltage and current.

3.4 Conclusion

The grid model is based on the incidence matrix T to connect the nodes in a radial grid. This flexibility

allows to model each radial grid topology by adjusting this incidence matrix. In this paper, the example of the IEEE 34 node test feeder is used. From the household power demand and the PV power production, it can be calculated how much extra power is needed from the grid or has to be put on the grid. With this, the nodal and line currents and nodal voltages can be calculated with Modelica. As such, it is possible to examine the influence on the voltage profile of the grid.

4 A residential electricity grid with variable density of PV integration

4.1 Load profiles of households

In this paper a distribution grid for a residential district is simulated. Each household has a specific load profile. Synthetic residential load profiles for households are provided by the Flemish Regulator for the Electricity and Gas market (VREG). A set of 16 different load profiles which cover each one day are available. These load profiles present the average load profiles of the Flemish population. These are corrected to obtain an average yearly electricity consumption in the order of 5,000 kWh. The dependency of the consumption upon outdoor temperature changes are given, but is however for simplicity not taken into account since the purpose of this paper is only to give an example of a grid application with a household and PV connection. Each profile gives the power demand on a 30-minute basis as shown in Figure 4.

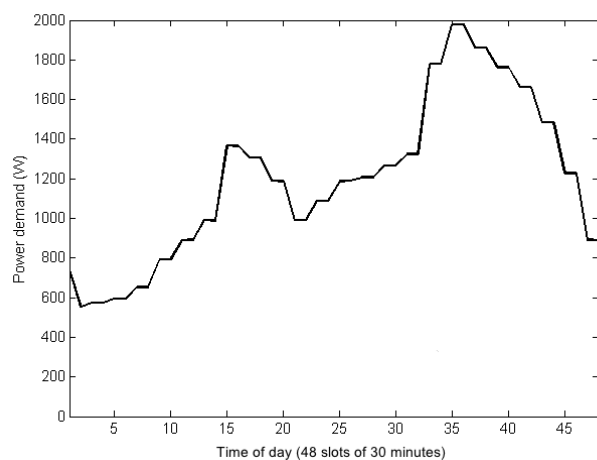


Figure 4: One of the 16 load profiles of an average Flemish household

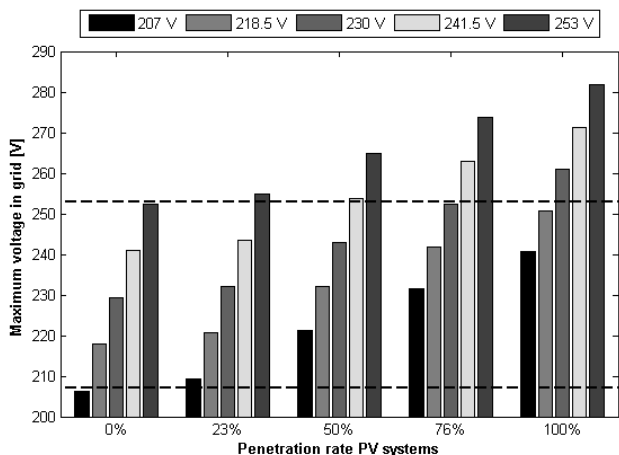


Figure 5: Maximum voltage in the grid at different PV system densities and fixed voltage at the transformer

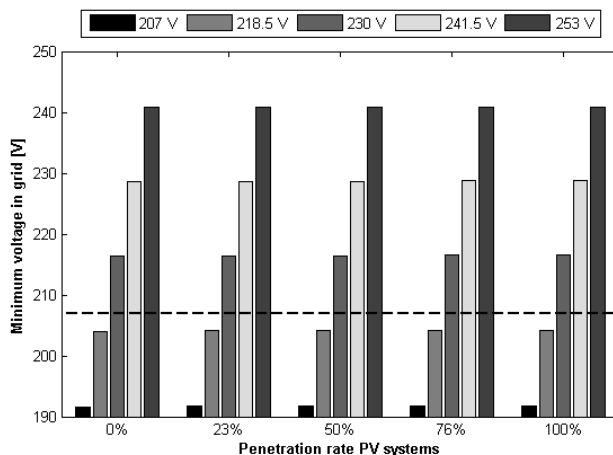


Figure 6: Minimum voltage in the grid at different PV system densities and fixed voltage at the transformer

4.2 Simulation parameters

It is supposed that at each node in the grid there is a household with a load profile. The 16 available load profiles are randomly assigned to each household.

A PV system can be installed for each household. The PV system consists of 30 panels resulting in a power of 6 kW_p . One goal of this paper is to evaluate the effect of the integration of PV systems on a distribution grid. The integration of PV systems¹ is evaluated for different penetration rates, namely a penetration of 0%, 17%, 23%, 32%, 50%, 67%, 76%, 82% and 100% of the nodes [7].

The reference scenario is a distribution grid with 0% PV systems. The source of the grid will be adapted to $230 \pm 5\% \text{ V}$ and $230 \pm 10\% \text{ V}$. A distribution system operator needs to keep the voltages at each node within certain voltage limits ($230 \pm 10\% \text{ V}$ for 95% of the time). To this end, the transformer tap will be changed to change the source voltage of the grid. E.g. when no PV systems are available, the voltage at the source has to be higher to compensate for lower voltages due to voltage drops in the feeders, caused by the power flow towards the household. When on the other hand PV systems produce power which is put back on the grid, these PV systems will lift the voltages by sending power back into the grid, thus the voltage at the transformer needs to be lower in this case (the transformer tap setting² has to be changed). This voltage lift will depend on the amount of power sent back to the grid.

¹All PV systems are oriented towards the south and have a 34° inclination angle of the panels.

²A transformer only has a discrete number of taps.

4.3 Results

In this paragraph some simulation results are shown. The idea is to clarify the problem that might occur when implementing a large amount of PV systems into a residential grid, since the voltages in the grid need to be kept between certain voltage limits ($230 \pm 10\% \text{ V}$).

Figure 5 shows the influence from leaving the transformer tap at a higher setting when PV systems are installed. It can be seen that overvoltages would occur at a PV system integration of 23% for a tap setting of $230 + 10\% \text{ V}$ and at an integration of 50% for a tap setting of $230 + 5\% \text{ V}$.

In contrast with this, Figure 6 shows undervoltage problems in the grid when the tap has been set on a low setting. It can be seen that the amount of PV integration has no effect on the minimum voltage in the grid. This can be explained by the consumption peaks that occur when there is no production from the PV panels, i.e. during a very cloudy period or during the evening. This shows that setting the transformer tap at a lower setting than 230 V is not a solution to the problem, since undervoltages will occur.

Generally tap settings will be made to anticipate only on load. This means that the voltage at the transformer will be kept higher than nominal to ensure it does not drop below 207 V anywhere. When there are only loads in the grid, the voltage at the transformer station will be the highest of the grid. The line at the bottom of Figure 7 shows this grid voltage profile without local generation (no PV systems). Figure 7 shows the voltage profile at the nodes indicated by the red line in Figure 8 at a time with high local electricity generation.

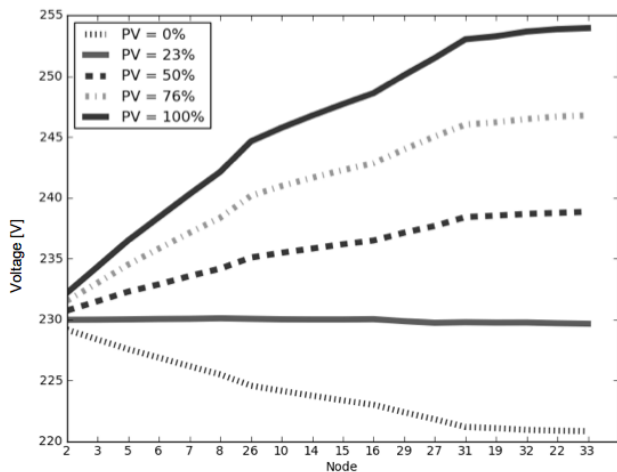


Figure 7: Voltage profile on a sunny day (at a time with high local electricity generation) in a 34 bus grid with different PV system penetration rates

The voltage profile in Figure 7 is visualized for different penetration rates for PV systems in the distribution grid. It is clear from the figure that when there is some PV penetration the voltage at the transformer is no longer the lowest in the grid. A closer look at the voltage profile for a PV density of 23% shows that the voltage profile is fluctuating around 230 V. This kind of profile is more difficult to control by adjusting the tap settings, because the voltage at different nodes needs to be known. For higher penetrations of PV systems, the grid voltage at the nodes at the end of the grid are much higher than the voltage at the transformer. At some nodes the slope of the voltage profile changes since some parts of the grid are not included in this figure, but still these nodes have their influence on the voltage profile for the rest of the grid.

5 Conclusions

This paper shows that object-oriented modelling in Modelica can be an attractive alternative for modelling electrical grids with power simulations.

A model for photovoltaic systems is described and implemented in Modelica, with an emphasis on the preservation of flexibility. Moreover, an electrical grid model is presented in Modelica. The grid model is structured around the incidence matrix \mathbf{T} and the impedance matrix \mathbf{Z} . This approach ensures optimal compatibility and flexibility with existing test feeder topologies.

Finally, an example case shows how these two

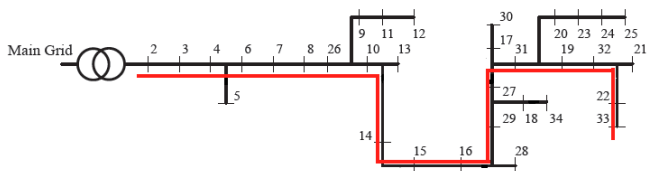


Figure 8: Grid topology IEEE 34 node test feeder indicating the nodes shown in the voltage profile of Figure 7 (red line)

models can be combined. A simulation of the IEEE 34 bus node test feeder with standard household load profiles and different degrees of PV density is shown and discussed regarding possible voltage problems in the grid. A high penetration of local generation by means of PV systems has a high influence on the grid voltage during sunny days. This is demonstrated for PV systems of 6 kW_p for different penetration rates. To prevent overvoltages, the transformer tap has to be adjusted to have a lower source voltage during these time periods.

Multidisciplinary research will be facilitated by further developing models that can be easily manipulated and interconnected.

6 Future work

In future work, Modelica *records* will be made for different network layouts. IEEE test grids will be implemented, but also an extension could be made to implement in home electricity grids. Modelica *records* for the parameters of different PV panels will also be implemented and the PV inverter will be modeled in more detail.

The grid model itself will be extended to include the possibility to simulate a 3 phase distribution grid and to include the distribution grid transformer.

In the future other types of distributed generation (e.g. small wind turbines) or storage (e.g. batteries to store a surplus of electricity from the PV panels to avoid too high grid voltages) may also be implemented, together with advanced control strategies for demand side management.

7 Acknowledgments

The authors gratefully acknowledge the K.U.Leuven Energy Institute (EI) for funding this research through granting the project entitled *Optimized Energy Networks for Buildings*.

References

- [1] R. De Coninck, R. Baetens, B. Verbruggen, J. Driesen, D. Saelens, and L. Helsen, "Modelling and simulation of a grid connected photovoltaic heat pump system with thermal energy storage using Modelica," in *8th International Conference on System Simulation in Buildings*, 2010, pp. 1-21.
- [2] W. Kersting, "Radial Distribution Test Feeders," in *IEEE Power Engineering Society Winter Meeting*, vol. 2, Columbus, Ohio, USA, 2001, pp. 908-912.
- [3] W. De Soto, S. Klein, and W. Beckman, "Improvement and Validation of a Model for Photovoltaic Array Performance," in *Solar Energy*, vol. 80, 2006, pp. 78-88.
- [4] D. Sera, R. Teodorescu, and P. Rodriguez, "PV panel model based on datasheet values," in *IEEE International Symposium on Industrial Electronics*, 2007, pp. 2392-2396.
- [5] Meteotest, 2008. METEONORM Version 6.1 - Edition 2009.
- [6] K. Clement-Nyns, E. Haesen, and J. Driesen, "The Impact of Charging Plug-In Hybrid Electric Vehicles on a Residential Distribution Grid," in *IEEE Transactions on Power Systems*, vol. 25, 2010, pp. 371-380.
- [7] E. Haesen, J. Driesen, and R. Belmans, "A Long-Term Multi-objective Planning Tool for Distributed Energy Resources", in *IEEE PES Power Systems Conference & Exposition*, Atlanta, Georgia, USA, Oct.29-Nov.1, 2006, pp. 741-747.

An Open Source Modelica Graphic Editor Integrated with Electronic Notebooks and Interactive Simulation

Syed Adeel Asghar¹, Sonia Tariq¹, Mohsen Torabzadeh-Tari¹, Peter Fritzson¹, Adrian Pop¹, Martin Sjölund¹, Parham Vasaiely², Wladimir Schamai²

¹PELAB – Programming Environment Lab, Dept. Computer Science
Linköping University, SE-581 83 Linköping, Sweden

²EADS Innovation Works, Engineering & Architecture, 21129 Hamburg, Germany
adeel.asghar@liu.se, x10sonta@ida.liu.se, {mohsen.torabzadeh-tari,peter.fritzson,adrian.pop,martin.sjolund}@liu.se,
Parham.Vasaiely@gmx.de, Wladimir.schamai@eads.net

Abstract

This paper describes the first open source Modelica graphic editor which is integrated with interactive electronic notebooks and online interactive simulation.

The work is motivated by the need for easy-to-use graphic editing of Modelica models using OpenModelica, as well as needs in teaching where the student should be able to interactively modify and simulate models in an electronic book. Models can be both textual and graphical. The interactive online simulation makes the simulation respond in real-time to model changes, which is useful in a number of contexts including immediate feedback to students.

Keywords: Graphic editing, notebook, teaching, interactive, Modelica, modeling, simulation, online

1 Introduction

OMEdit, the OpenModelica Connection Editor, is the new Graphical User Interface for graphic model editing in OpenModelica. It is implemented in C++ using the Qt 4.7 graphical user interface library, and supports the Modelica Standard Library version 3.1 that comes with the OpenModelica installation.

OMEdit provides a user friendly environment for:

- *Modeling* – Easy Modelica model creation.
- *Pre-defined Models* – Browsing the Modelica Standard library to access the provided models.
- *User defined models* – Users can create their own models for immediate usage and later refinement and reuse.
- *Component Interfaces* – Smart connection editing for drawing and editing connections between model interfaces.

- *Simulation subsystem* – Subsystem for running simulations (not online) and specifying simulation parameters start and stop time, etc.
- *Online Simulation* – Online interactive simulation where the simulation responds in real-time to user input and changes to parameters.
- *Plotting* – Interface to plot variables from simulated models.
- *OMNotebook integration* – being able to open a graphical connection diagram in an electronic notebook, edit it, and paste it back.

OMEdit uses the OmniORB CORBA implementation to communicate with the OpenModelica Compiler.

Modelica 3.2 Graphical Annotations are interpreted for drawing Modelica Standard Library component models and user defined models. As a result, the interoperability with other Modelica tool vendors becomes easier as the Modelica icon and diagrams defined in other tools supporting the Modelica 3.1 or Modelica 3.2 standards are easily handled in OMEdit. The annotations are also used for displaying Modelica documentation in OMEdit.

1.1 Structure of the Paper

Section 3 describes the usage of OMEdit and also demonstrates how a `DCmotor` model is created using OMEdit. Section 4 explains the OMEdit communication process with OMC through the CORBA interface. How user defined and Modelica component model shapes are created through annotations is discussed in section 5.

Section 6 elaborates the interactive simulation mechanism that is still under development in OMEdit. Section 7 briefly describes how OMEdit can interact with OMNotebook and how users can launch electronic

notebooks in OMEdit. Moreover, Section 8 presents related work and in the end, Section 9 suggests some future work.

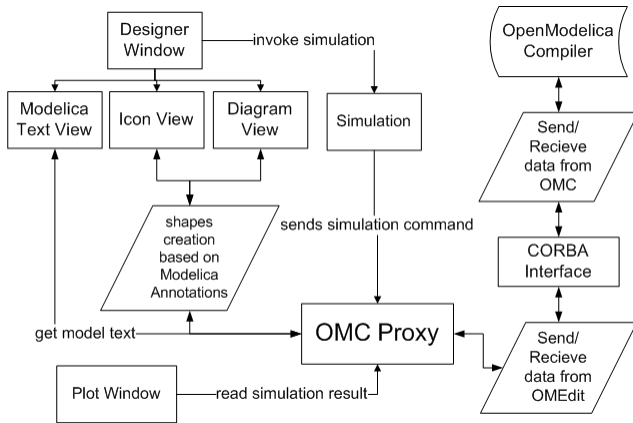


Figure 1. OMEdit High-level View.

2 Requirements and Motivation

This work is motivated by the need for easy-to-use graphic editing of Modelica models using OpenModelica, as well as needs in teaching where the student should be able to interactively modify and simulate textual and graphical models in an electronic notebook.

The interactive online simulation makes the simulation respond in real-time to model changes, which is useful in a number of contexts, especially teaching where immediate feedback to students enhances the effectiveness of learning.

A recently developed interactive learning material called DrControl is depicted in Figure 2.

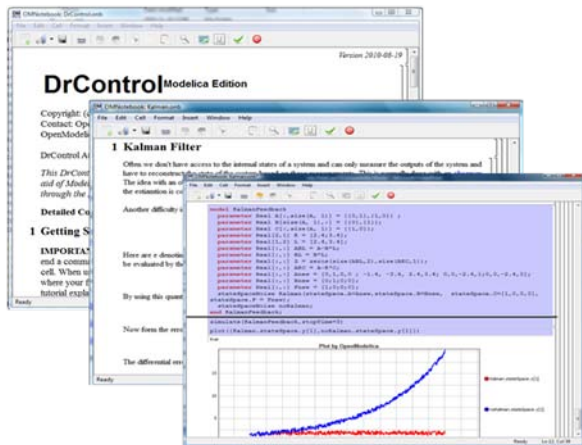


Figure 2. DrControl for teaching control theory with Modelica.

DrControl is a new active electronic notebook course material based on OMNotebook for teaching control theory and modeling with Modelica, including graphic connection diagrams supported by OMEdit. It contains explanations about basic concepts of control theory

along with Modelica exercises. Observer models, Kalman filters, and linearization of non-linear problems are some of the topics in the course used in control of a pendulum, a DC motor, and a tank system model among others.

3 Using OMEdit

This section gives a brief introduction about how to use OMEdit and also demonstrates how to create a DCmotor model.

3.1 Introductory Model in OMEdit

Since Modelica is an equation-based language and OMEdit is a connection editor, we will for a small introductory model demonstration in OMEdit show how a DCmotor model is created in OMEdit.

3.1.1 Creating a new file

Creating a new file/model in OMEdit is rather straightforward. In OMEdit the new file can be of type model, class, connector, record, block, function or package. The user can create any of the model types mentioned above by selecting File > New from the menu. Alternatively, you can also click on the drop down button beside new icon shown in the toolbar right below the File menu. See Figure 3.

In this introductory example we will create a new model named DCmotor. By default the newly created model will open up in the tabbed view of OMEdit, also called Designer Window, and become visible.

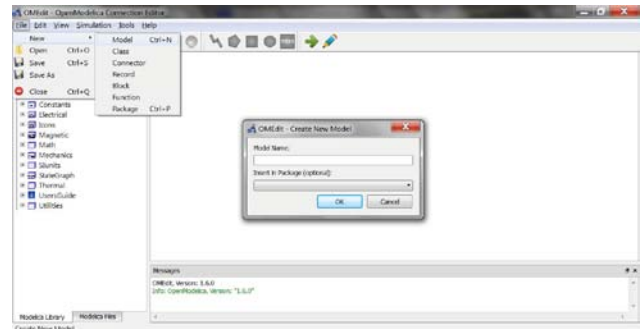


Figure 3. Creating a new file/model.

All the models are created in the OMC global scope unless the user specifies the parent package for it.

3.1.2 Adding Component Models

The Modelica Standard Library is loaded automatically and is available in the left dock window. The library is retrieved through the loadModel(Modelica) API function and is loaded into the OMC symbol table and workspace after the command is completed.

Instances of the component models available in the Modelica Standard Library can be added to the currently edited model by doing a drag and drop from the Library Window. Navigate to the component model in the library tree, click on it, drag it to the model you are building while pressing the mouse left button, and drop the component where you want to place it in the model.

For this example we will add four components as instances of the models Ground, Resistor, Inductor and EMF from the Modelica.Electrical.Analog.Basic package, an instance of the model SignalVoltage from the Modelica.Electrical.Analog.Sources package, one instance of the model Inertia from the Modelica.Mechanics.Rotational.Components package and one last instance of the model Step from the Modelica.Blocks.Sources package.

3.1.3 Making Connections

In order to connect one component model to another the user simply clicks on any of the ports. Then it will start displaying a connection line. Then move the mouse to the component where you want to finish the connection and click on the component port where the connection should end. You do not need to hold the mouse left button down for drawing connections.

In order to have a functioning DCmotor model, connect the Resistor to the Inductor and the SignalVoltage, EMF to Inductor and Inertia, Ground to SignalVoltage and EMF, and finally Step to SignalVoltage. Check Figure 4 to see how the DCmotor model looks like after connections.

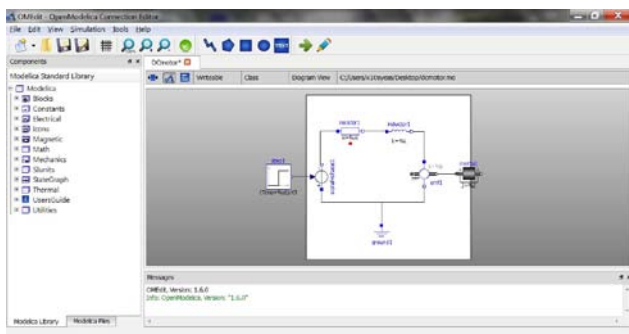


Figure 4. DCmotor model after connections.

3.1.4 Simulating the model

OpenModelica models are simulated using the simulate command of OMC. The simulate command has following parameters;

- Simulation Interval
 - Start Time
 - Stop Time

- Output Interval
 - Number of Intervals
 - Output Interval
- Integration
 - Method
 - Tolerance
 - Fixed Step Size

The OpenModelica Connection Editor provides an easy interface for simulation of models and allows the user to fill in the parameters before starting the simulation process.

The OMEdit Simulation dialog can be launched either from Simulation > Simulate or by clicking the simulate icon from the toolbar. Once the user clicks on simulate! button, OMEdit starts the simulation process, at the end of the simulation process the Plot Variables window, useful for plotting, will appear at the right side. Figure 5 shows the simulation dialog.

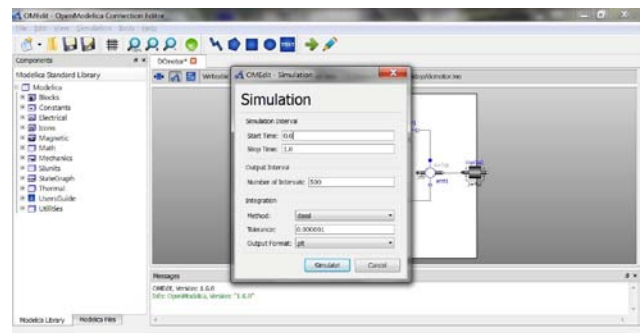


Figure 5. Simulation Dialog.

3.1.5 Plotting Variables from Simulated Models

The instance variables that are candidates for plotting are shown in the right dock window. This window is automatically launched once the user simulates the model; the user can also launch this window manually either from Simulation > Plot Variables or by clicking on the plot icon from toolbar. It contains the list of variables that are possible to use in an OpenModelica plot. The plot variables window contains a tree structure of variables; there is a checkbox beside each variable. The user can launch the plotted graph window by clicking the checkbox.

Figure 6 shows the complete DCmotor model along with the list of plot variables and an example plot window.

4.4 What to do with the CORBA IOR File?

The IOR File contains the CORBA object reference as a string. The CORBA object is created by reading the string written in the IOR File. Here is an example with Qt C++ source code for starting OMC from OMedit and creating a CORBA object:

```
// create a unique file name
QString fileIdentifier;fileIdentifier =
qApp-
>sessionId().append(QTime::currentTime().t
oString().remove(":"));

QStringList parameters;
parameters << QString("+c=").append(this-
>mName).append(fileIdentifier) <<
QString("+d=interactiveCorba");

// start the OMC process
QProcess *omcProcess = new QProcess();
omcProcess->start( omcPath, parameters );

// read the file created by omc.exe
QFile objectRefFile (path_to_IOR_File);
int argc = 2;
static const char *argv[] = { "-
ORBgiopMaxMsgSize", "10485760" };
CORBA::ORB_var orb = CORBA::ORB_init(argc,
(char **)argv);
objectRefFile.open(QIODevice::ReadOnly);
char buf[1024];

// read the IOR string
objectRefFile.readLine( buf, sizeof(buf)
);
QString uri( (const char*)buf );

// create CORBA object
CORBA::Object_var obj = orb-
>string_to_object(uri.trimmed().toLatin1()
);
```

4.5 OMC API Enhancements

During the development of OMedit several issues with the OMC Application Programming Interface (API) were discovered:

- Annotations for some models could not be retrieved via `getIconAnnotation`, `getDiagramAnnotation` or `getDocumentationAnnotation`.
- `addConnection` and `updateComponent` did not work correctly.
- `renameComponent` was very slow.
- The package `Modelica.UsersGuide` does not have any icon/diagram annotation but it has a non-standard Dymola annotation.

For example `getIconAnnotation(Modelica.Electrical.Analog.Resistor)` did not work because the `Resistor` model had component references inside the annotations. This problem was solved by symbolically elaborating (instantiating) the `Resistor` model, con-

stant evaluating the `useHeatPort` parameter, and then elaborating the annotation record with this constant value.

Using constant evaluated parameters from elaborated model does not work for annotations that contain `DynamicSelect` and additional support for such annotations is needed. Unfortunately the `DynamicSelect` annotation creates problems for Modelica software that uses a client-server paradigm since it connects an annotation with a simulation, not with the actual model. However, `DynamicSelect` can still be handled by returning the entire expression to the client (here OMedit) which could link a simulation variable to the annotation.

Retrieving the documentation annotation for MSL 3.1 did not work at first because these annotations had been moved (MSL 2.x had no such requirements) to the end of the class definitions (typically in an equation section) and OMC only searched the public sections. This was solved easily in OMC by searching the entire model for the documentation annotation.

To make it easier to find which annotations cannot be retrieved correctly OMC was changed to return the exact annotation that was present in the model. Using this feature the problematic parts of the communication between OMedit and OMC was debugged.

Updating components and adding connections to classes had small issues that were fixed to support OMedit.

The package `Modelica.UsersGuide` and several others do not have any icon/diagram annotation. Displaying these packages in the MSL 3.1 browsing tree did not look nice. However, we observed that these packages has a non-standard Dymola specific annotation which is: `__Dymola_DocumentationClass = true`. In order to retrieve this annotation in OMedit the OMC API had to be extended with a new function: `getNamedAnnotation(Modelica.UsersGuide) => true`. Now these packages can display a predefined icon in the tree browser.

To automatically test which component models have problems a script was written in OMedit that walks the entire MSL 3.1 and calls OMC API functions on these models to see if the retrieved information was correct or not. A list with problematic models was built. Subsequently these issues were solved one-by-one.

The function to rename a component, `renameComponent` API function, was extremely slow when MSL 3.1 was loaded. This occurred because OMC had to go through all models and components and do a renaming refactoring. To resolve this and provide a faster functionality, we added a new API `renameComponentIn-`

Class that only renames the component locally in the model that is built using OMEdit and not in any other.

5 Annotations

Modelica annotations are used for storing auxiliary information about a model such as graphics, documentation or versioning etc. [2]. Once OMEdit is connected with OMC it can request the annotations. OMEdit uses three types of annotations;

- Annotations for Graphical Objects.
- Annotations for Connections.
- Annotations for Documentation.

5.1 Shapes/Component Models Annotations

All the shapes drawn in OMEdit are based on Modelica Annotations version 3.2. Graphical Annotations consist of two abstraction layers: the icon layer and the diagram layer. The icon layer contains the icon representation of a component and the diagram layer shows the inheritance hierarchy, connections, and inherited component models.

For example, a graphical icon representation of a Ground component model will look like this:

```
{-100.0, -
100.0,100.0,100.0,true,0.1,2.0,2.0,{Line(t
rue,{0.0,0.0},0,{{-
60,50},{60,50}},{0,0,255},LinePattern.Soli
d,0.25,{Arrow.None,Arrow.None},3,Smooth.No
ne),Line(true,{0.0,0.0},0,{{-
40,30},{40,30}},{0,0,255},LinePattern.Soli
d,0.25,{Arrow.None,Arrow.None},3,Smooth.No
ne),Line(true,{0.0,0.0},0,{{-
20,10},{20,10}},{0,0,255},LinePattern.Soli
d,0.25,{Arrow.None,Arrow.None},3,Smooth.No
ne),Line(true,{0.0,0.0},0,{0,90},{0,50}},
{0,0,255},LinePattern.Solid,0.25,{Arrow.No
ne,Arrow.None},3,Smooth.None),Text(true,{0
.0,0.0},0,{0,0,255},{0,0,0},LinePattern.So
lid,FillPattern.None,0.25,{-144,-
19},{156,-
59}},{ "%name", 0,TextAlignment.Center)}}
```

This graphical representation of the Ground model is parsed by OMEdit for drawing this component model. The icon annotation is retrieved from OMC through the `getIconAnnotation` API command. Each graphical object is built up using the primitive graphical types; Line, Polygon, Rectangle, Ellipse, Text and Bitmap [2].

The primitive graphical types in OMEdit are handled through the `QGraphicsItem` class of Qt. A `ShapeAnnotation` class was created which is derived from `QGraphicsItem` and `QObject`. This class is an abstract class which contains classes of all primitive graphical elements.

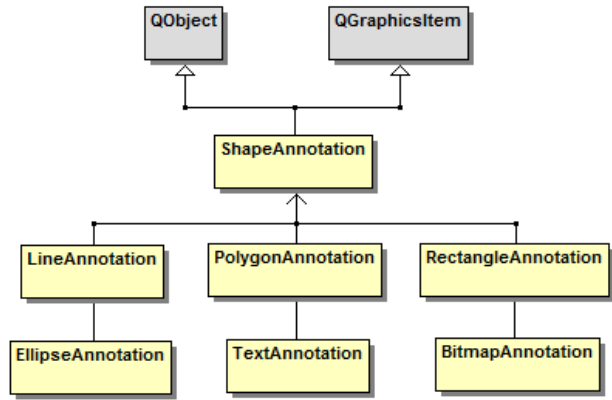


Figure 8. Classes hierarchy for predefined graphical elements.

5.2 Connection Annotation

The connection annotation defines the graphical representation of a connection between two component models. An example of connection annotation string is:

```
connect (a.x, b.x)
annotation(Line(points={{-25,30}, {10,30},
{10, -20}, {40,-20}}));
```

The connection annotation is composed of the primitive graphical type `Line`. The points of the line define the connection line co-ordinates between two connecting component models.

OMEdit creates an object of `Connector` class for each connection. Each `Connector` contains instances of `ConnectorLine` depending on the number of points in a connection. The `Connector` class is derived from `QGraphicsWidget` class which is container class for graphical objects. The `ConnectorLine` class is derived from `QGraphicsLineItem` which represents a single line. If we have n points in a connection annotation then we have $n-1$ instances of `ConnectorLine`. In short n number of points creates $n-1$ lines. The following shows the implementation of connection annotation in OMEdit.

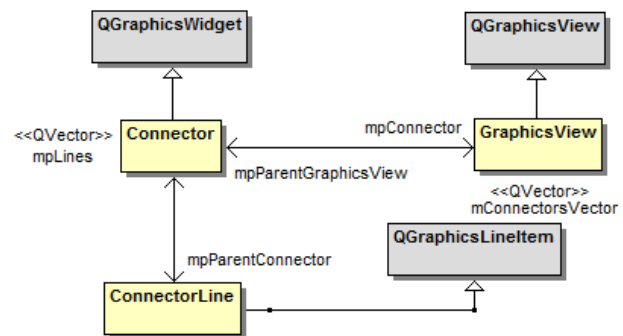


Figure 9. Implementation of connection annotation.

5.3 Documentation Annotation

The documentation annotation is used for textual descriptions of models. It is written as follows:

```
documentation_annotation:
annotation(" Documentation "(" "info" "="
STRING
["," "revisions" "=" STRING ] )" ")"
```

OMEdit requests OMC for the documentation of a specific component/library through the `getDocumentation` command and OMC returns the `info` annotation contained inside the documentation annotation which is a string. The tags `<HTML>` and `</HTML>` define the start and end of the string.

The `QWebView` class of Qt is used for displaying the HTML string of documentation annotation. The HTML string contains four types of links:

- Hyperlinks – Used to navigate to external websites.
- Image Links – Used to reference the local image files.
- Modelica Links – Used for linking to other component models.
- Mailto Links – Used to display email addresses that can be used for future contacts.

`QWebView` has built-in support for images so we didn't have to handle that. We just set the proper base path where all the images were located. However, for hyperlinks and mailto links we used the `QDesktopServices` class. This class uses the default system browser in case of hyperlink and default email client in case of mailto link. The Modelica links are special links which starts with `Modelica://` and reference to some component model or a package. Figure 10 shows the implementation of documentation annotation in OMEdit.

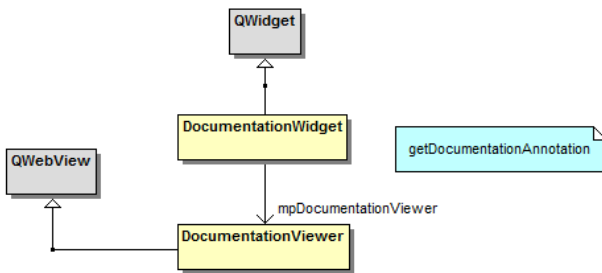


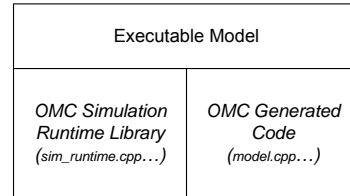
Figure 10. Implementation of documentation annotation.

6 Interactive Simulation

In order to offer a user-interactive and time synchronous simulation, OpenModelica has an additional subsystem to fulfill general requirements on such simulations, OpenModelica Interactive (OMI), shown in Fig-

ure 11. With OMI the user will be able to stimulate the system and interacting with it at runtime.

After creating and elaborating a Modelica model it is possible to simulate the model with OpenModelica. The outcome of calling the `simulate` or `buildModel` operation from the interactive session handler, is an executable, standalone C/C++ program generated from the internal simulation runtime code and the generated C/C++ model code by OMC (in this case `model.cpp`).



This executable contains the full Modelica model translated to C/C++ code based on all required equations, conditions and including different solvers. It offers both a non-interactive as well as an interactive simulation facility.

Since version 1.5.0 OpenModelica has an additional subsystem in order to offer a user-interactive and time synchronous simulation. This module is part of the simulation runtime core and is called “OpenModelica Interactive” (OMI). As mentioned above OMI will result in an executable simulation application, such as the non-interactive simulation. The following are some general functionalities of an interactive simulation runtime:

- The user will be able to stimulate the system during a running system simulation and to observe its reaction immediately.
- The simulation runtime behavior will be controllable and adaptable to offer an interaction with a user.
- A user will receive simulation results online during a simulation synchronous to real time, neglecting network process time and some other factors like scheduling of processes from the operation system.
- In order to offer a stable simulation, a runtime function will inform the user interface of errors and consequential simulation aborts.
- Simulation results will not under-run or exceed a tolerance compared to a thoroughly reliable value, for a correct simulation.
- Communication between a simulation runtime and a user interface will use a well defined interface and be based on common technology, in this case network communication.

In this case the `simulate` operation cannot be used. Instead the `buildModel` operation is needed.

To start an interactive simulation there is a need for more information, such as network configurations.

An important modification/addition to the semantics of the Modelica language during interactive simulation is the fact that parameters are changeable while simulating interactively using OMI. All properties using the prefix `parameter` can be changed during an interactive simulation. The fully qualified name is used as a unique identifier, so a parameter value can be found and changed regardless of its hierarchical position in the model. For more information see the OpenModelica System Documentation [1].

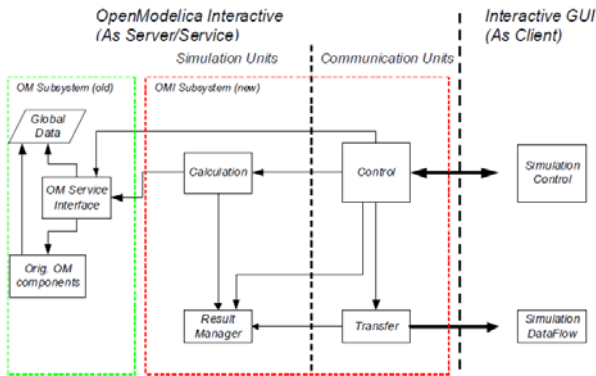


Figure 11. OpenModelica interactive system architecture overview.

7 Interaction with OMNotebook

OMEdit provides an environment where connection diagrams can be integrated with electronic interactive notebook. The idea is that the user performs the modeling in the connection editor and can subsequently export his/her models to an electronic notebook.

Alternately, the model in an electronic notebook is just an image. The model including its equations, algorithms, annotations etc. are hidden behind the picture. Thus, OMEdit is integrated with the OMNotebook tool [7], allowing users to click on the image and launch the model in connection editor where user can manage the connections, add/remove component models etc.

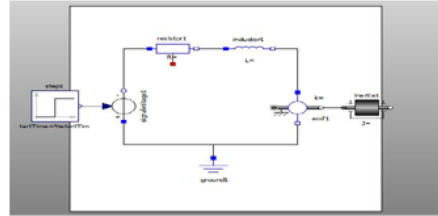
Figure 12 shows an electronic notebook with a `DCmotor` model as an image. When the user double clicks on the image, an OMEdit editing view is popped up, allowing both textual and graphical editing.

Exercise - Graphical Modeling

1 The DC Motor

A) DC Motor

Make a simple DC-motor using the Modelica standard library that has the following structure:



model ...

Figure 12. OMEdit integrated with OMNotebook used in a teaching material with exercises.

8 Related Work

There is previously one open source graphical editor available for OpenModelica:

- *SimForge* – Graphical and Textual Open Source Model Editor by Politecnico di Milano [3].

We have tried this editor for teaching, but found that the current implementation is too slow, not stable enough, and does not integrate with OMNotebook and interactive simulation.

There are also several commercial tools available for graphical modeling, e.g.:

- *Dymola* – Developed by Dynasim. Dymola, Dynamic Modeling Laboratory, is a complete tool for modeling and simulation of integrated and complex systems for use within automotive, aerospace, robotics, process and other applications [4].
- *MathModelica* – Developed by MathCore Engineering AB. MathModelica is a powerful, flexible and extensible system for multi-engineering modeling and simulation [5].
- *MapleSim* – High Performance Physical Modeling and Simulation from Maplesoft [6].

These are professional products that work well, but are not freely available, and are not open source. Also, they are typically not integrated with electronic books. An earlier version of MathModelica was integrated with the Mathematica electronic book, but did not provide interactive online simulation. The electronic notebook from Maplesoft is Maple-based and is pure textual. Also the Modelica language support is lacking in this tool.

9 Future Work

The first version of OMEdit is part of the OpenModelica 1.6 release. The version integrating OMEdit and Interactive simulation with OMNotebook will be available very soon, probably in the 1.6.1 release.

Moreover, somewhat improved 2D plotting is currently on the way. Future enhancements on the wish list include improved 3D graphic animation and support for displaying inheritance dependencies and sources of inherited equations and declarations.

10 Acknowledgements

This work has been supported by EU project Lila and Vinnova in the ITEA2 OPENPROD project. The Open Source Modelica Consortium supports the OpenModelica work.

References

- [1] Adeel Asghar and Sonia Tariq. *Design and Implementation of a User Friendly OpenModelica Connection Editor*, master thesis LIU-IDA/LITH-EX-A-10/047-SE, Linköping University, Sweden, 2010.
- [2] Open Source Modelica Consortium. *OpenModelica System Documentation Version 1.6*, November 2010. <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] SimForge. <http://trac.ws.dei.polimi.it/simforge/>.
- [5] Dymola. *Dynamic modeling tool*, <http://www.dynasim.se>.
- [6] MathModelica. <http://www.mathcore.com/products/mathmodelica/>.
- [7] Peter Fritzson, Johan Gunnarsson, Mats Jirstrand. MathModelica - An Extensible Modeling and Simulation Environment with Integrated Graphics and Literate Programming. In *Proceedings of the 2nd International Modelica Conference*, March 18-19, 2002, Munich, Germany.
- [8] Anders Fernström, Ingemar Axelsson, Peter Fritzson, Anders Sandholm, Adrian Pop. OMNotebook – Interactive WYSIWYG Book Software for Teaching Programming. In Proc. of the Workshop on Developing Computer Science Education – How Can It Be Done? Linköping University, Dept. Computer & Inf. Science, Linköping, Sweden, March 10, 2006.

Functional Digital Mock-up and the Functional Mock-up Interface – Two Complementary Approaches for a Comprehensive Investigation of Heterogeneous Systems

Olaf Enge-Rosenblatt, Christoph Clauß, André Schneider, Peter Schneider
 Fraunhofer Institute for Integrated Circuits, Division Design Automation
 Zeunerstraße 38, 01069 Dresden, Germany
 Olaf.Enge-Rosenblatt@eas.iis.fraunhofer.de

Abstract

Functional Digital Mock-up (FDMU) and Functional Mock-up Interface (FMI) are two keywords arising in the last years in simulation technology. In this paper, we would like to show that both principles, aiming at a comprehensive investigation of heterogeneous systems, e.g. from mechatronics, are not necessarily competing with each other but may be combined to benefit from the ideas behind. The approaches are based on different ideas and cover different aspects of the interaction of modern simulation tools. For that reason different constraints have to be considered, which do not make things easier. Both principles have advantages and disadvantages. However, by combining both ways, a powerful framework for handling a broad variety of simulation tasks can be formed. In the paper, a possible approach for integrating both technologies will be shown.

Keywords: *FDMU; functional digital mock-up; FMI; functional mock-up interface; co-simulation; heterogeneous system; simulation algorithm*

1 Introduction

In today's industry, the product development process is more and more characterized by intensive usage of simulation. But in all branches, a large variety of languages, simulators, and environments are used which are incompatible to each other in the most cases. Hence, the task of coupling different simulation tools, e.g. by co-simulation concepts, has been moving increasingly into the focus in the last years.

In this context, many activities within the simulation community have been able to be recorded concerning the integration of the principle of *Functional Mock-up* into the world of simulation and simulators. The main target of these attempts is to combine the ideas of the digital mock-up processes with functional aspects. Even though both approaches are open for

different modeling languages and simulators, the general purpose modeling language Modelica plays an important role on both sides.

Two "main streams" can be distinguished: the FDMU approach [2][7][9] and the FMI approach [3][5][6]. The FDMU ideas are driven by four German Fraunhofer institutes funded by the Fraunhofer-Gesellschaft. In contrast, the FMI ideas are favored by a network of companies and research institutions spread widely across Europe and working together within the MODELISAR project funded within the ITEA2 framework. Both approaches have advantages and disadvantages and both deal with general ideas of co-simulation. While the FDMU approach focuses on the combination of different simulators with a shared and interactive visualization, the FMI consortium thinks about model exchange and co-simulation ideas. More information about both approaches and a comparison of architectures and data flows is given in the following sections.

The Fraunhofer Institute for Integrated Circuits, Division Design Automation in Dresden has been taken part in both projects. That's why the idea to investigate some opportunities of a combination of both principles is not very far. Taking the differences of both principles into account possible ways to combine both approaches are shown in this paper.

2 FDMU

The main goals of the Fraunhofer-internal project Functional DMU were, first, to develop a principle of a general, tool-independent, and web service-based framework for coupling different simulation tools with each other as well as, second, to provide an integrative (and simultaneously interactive) visualization tool to unify the user's view. The output of the project is a fully implemented ready-to-use framework, the so-called FDMU framework, which can be used in connection with a large variety of simulation tools.

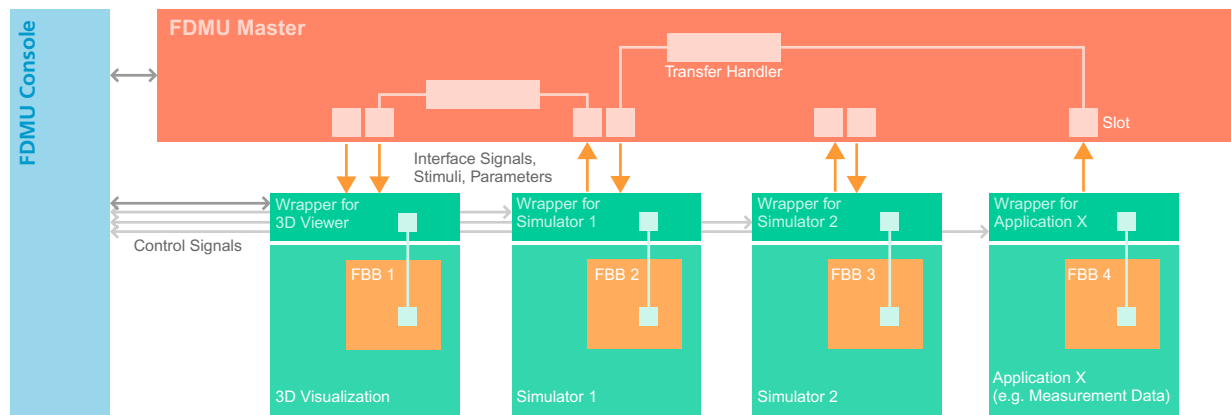


Figure 1: FDMU principle with Master Simulator

2.1 Basic idea

Functional Digital Mock-up (FDMU) is a substantial extension of the widely introduced DMU approach which is used for investigating geometrical and mechanical properties of mechanical systems. The FDMU approach combines traditional digital mock-up (i.e. geometrical information) and aspects of behavioral system description (i.e. functional information). An additional, new and very important point is the possible interaction between visualization and numerical simulation in both directions which is governed by a master simulator. This *FDMU Master Simulator* is also capable to accomplish the communication between different simulation tools. Hence, if dealing with a complex simulation task, more than one simulation tool can be incorporated to solve it.

Finally, the FDMU approach is predestined for multi-physical (or multi-domain) systems like often considered with appropriate modeling languages like Modelica.

2.2 Conceptual properties

In this paper, only the main concepts shall be presented. For more information please see the FDMU references, especially [2] and [7].

FBB and FSM

Within the FDMU approach, the concept of a so-called *Functional Building Block* (FBB) is proposed. An FBB is an envelope summarizing geometric information (CAD models), behavioral models (e.g. described by differential-algebraic equations), and communication interfaces into one basic data module. Geometric information and behavioral information have to be created within their particular modeling tools. These models remain in their associated

data files. Pointers to these files as well as all interface information and the mapping between geometrical data and the interface quantities of the behavioral model are collected within the FBB using the modeling language SysML for a unique description. Within every FBB, a simulator tool is defined, too, which is capable to simulate the FBB's behavioral part.

A complete *FDMU Simulation Model* (FSM) consists of one or more FBB. Every input of an FBB must have an appropriate output belonging to another FBB. Furthermore, outputs can be propagated to the visualization to show simulation results using e.g. a geometric 3D model or some kind of plot versus time.

Master Simulator

When simulating an FSM, different simulation tools have generally to interact with each other. This is realized by the concept of a flexible co-simulation. The governing instance, the FDMU Master Simulator, is used to ensure the correct communication between all involved simulators as well as the correct delivering of simulation results to the visualization tool. The Master Simulator is the centralized controlling software and, therefore, the main module of the FDMU approach. It does not contain a model itself. It controls the signal flows between all participating software modules.

Web services

One of the basic concepts of the FDMU approach is the usage of Web services for communication between FDMU Master Simulator, different simulator tools, and the visualization. This provides the opportunity to distribute the tools needed for a particular simulation task among different computers which have only to be connected via Internet. This

advantage must be paid with a little loss of coupling performance. But it is planned to enable a replacement of some of the Web services by direct TCP/IP socket connections to improve efficiency.

Wrappers

FDMU Wrappers are used to establish a connection between the different simulation tools and the FDMU Master Simulator. Every wrapper is a software module especially designed for a certain simulation tool to realize the interaction with the tool's environment. To this end, the simulator must provide the capability to include external functions into the simulation code. The wrapper is docked to the simulator like an external function and encapsulates the original simulator. This way a unique interface of every simulator is guaranteed.

Data Transfer

During a simulation run, all simulation tools needed to calculate the behavior of a complete FSM have to run in batch mode. The tools receive simulation control commands via pipe from the wrapper and perform the simulation. The time schemes for communication between the simulation tools and the Master Simulator may completely differ from each other. Hence, the FDMU Master Simulator supports a complex concept of data transfer. There are re-sampling procedures (transfer handlers, *Figure 1*), buffering schemes (slots, *Figure 1*), and an implicit synchronization concept performed by the data flow. The underlying concept is that all data transfers are initiated by the simulator tools.

Visualization

The FDMU framework provides an interactive visualization tool (*Figure 1*) for presenting subsequently a moving 3D scene according to the currently calculated simulation results. It is also possible to interact via the 3D scene with the complete simulation process. This is a manifestation of the close combination of geometrical and functional descriptions. This feature enables the user to control the simulation process like starting, pausing and finishing the simulation as well as to change FBB parameters.

2.3 Realization

The Functional Digital Mock-up approach is implemented within the so-called FDMU framework which is available with different combinations of

components. The complete framework consists of a Master Simulator, some simulation tools encapsulated by wrappers, a visualization front end, and some data services. Wrappers for Dymola, Saber, Rhapsody, Simpack, and the multi-purpose tools Matlab and Simulink are currently available. Other wrappers could be realized and provided on request.

3 FMI

The goal of the ITEA2 project MODELISAR was the creation of a model-/ software-/ hardware-in-the-loop standard interface which is both tool and vendor neutral. This interface is called Functional Mock-up Interface (FMI). Via this interface, different tools will be able to communicate and act together to finally reduce both time and costs of development.

3.1 Basic idea

Starting with a technical system the components of which are treated by different simulation environments (simulators), the simulation of the whole system requires interaction of simulators. Taking into account that simulation is the solution of equations (DAE, PDE), the interaction can basically be managed in two ways:

Model exchange: The simulation environment of one component establishes the equations e.g. basing on a description language. The equations are passed over to another simulator which collects all component models and simulates all equations together. The results are then distributed.

Co-Simulation: Each component's simulator solves its own equations but values are exchanged which belong to more than one component. The simulators have to be synchronized.

Of course, hybrid forms of these two ways are possible.

The FMI standard is designed to support a wide variety of coupling possibilities. For each system it is open to decide which simulator should be used, and which way, model exchange or co-simulation, is to prefer. Once the FMI is established, the tool vendors are in charge to support it. Fitted with FMI, the tools should be able to interact.

3.2 Concept

For model exchange, the *Functional Mock-up Interface for Model Exchange* is defined. The intention is that the model exporting environment generates C code of a dynamic system model that can be

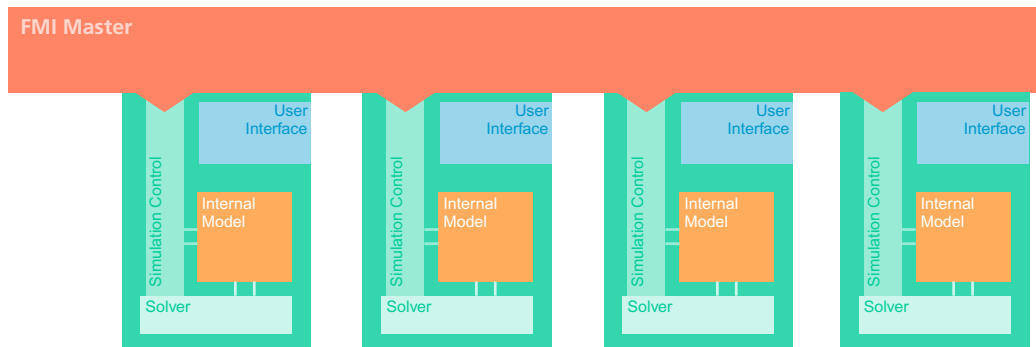


Figure 2: Principle of FMI for Co-Simulation

utilized by other (importing) simulation environments. Since models can describe differential, algebraic and discrete equations with time, state and step events, the interface is designed to be able to exchange all necessary values of variables (time, real, discrete values).

The Functional Mock-up Interface for Model Exchange consists of the *Model Interface* and the *Model Description Schema*. The Model Interface describes data types as well as functions compatible to a common basic mathematical description. All data and functions use the language C. The Model Description Schema contains all data which complete the model, but are not essential during model evaluation, e.g. numbers of variables, icons, documentation. A model specific zip-file contains all this information.

The C functions can either be provided in source form or in binary form, both of which can be added to the zip-file. The models are available by linking to the “importing” simulation environment. The interface is open to establish mechanisms for data transfer e.g. via the web.

The above mentioned ways (model exchange and co-simulation) are straightforward since the co-simulation approach needs the same data exchange like the model exchange approach, but needs additionally data which control the involved simulators. Therefore, the Functional Mock-up Interface for Model Exchange is a base for the Functional Mock-up Interface for Co-Simulation. Additional aspects come from solver coupling issues which are discussed in more detail in the following.

3.3 FMI for Co-Simulation

The *Functional Mock-up Interface for Co-Simulation* is a perspective standardized interface for coupling two or more simulation tools in a co-simulation environment. Co-simulation is a technique for

coupled time-continuous and time-discrete systems that exploits the modular structure of the coupled parts in all stages of simulation.

In co-simulation, different simulation tools have to interact while each of them has different properties concerning coupling algorithms. Important tool properties are:

- the ability to handle variable communication step sizes,
- the capability to discard and repeat communication steps,
- the capability to interpolate continuous inputs,
- the capability to provide information on a communication step (e.g. successfulness or error messages).

Otherwise, depending on the system to be simulated, there are different requirements to the simulation tool.

Therefore, the simulators of the components (slaves) are not directly coupled to each other but to a so-called master. The master’s tasks are:

- analyzing the connection graph,
- analyzing the properties of the involved slave simulators,
- choosing a dedicated master algorithm,
- forcing the slaves to initialize,
- forcing the slaves to simulate communication intervals,
- realizing the data transfer according to the connection graph as well as according to the chosen algorithm,
- termination of the slave processes.

The master algorithms will not be standardized. Master algorithms can be developed both as a separate tool as well as an included feature of an existing simulation tool which plays the role of the master.

The FMI for Co-Simulation is designed to support a large variety of master algorithms. Similar defined as for model exchange, the FMI for Co-Simulation consist of the *Co-Simulation Interface* and the

Co-Simulation Description Schema. The *Co-Simulation Interface* is a set of C functions for the exchange of input/output values as well as status information. One of the most important functions is the function

```
fmiDoStep(
    fmiComponent component,
    fmiReal currentCommunicationPoint,
    fmiReal communicationStepSize,
    fmiBoolean newStep);
```

which starts a slave identified by `component` to simulate one communication interval of the length `communicationStepSize` starting from the `currentCommunicationPoint`. Further functions are defined e.g. for retrieving the status information from the slave. The *Co-Simulation Description Schema* contains e.g. capability flags, which characterize the above mentioned properties of the involved slave simulator.

Besides pure technical aspects, issues of protection against unintentional know-how transfer and authorization of models are generally solved.

4 Comparing FDMU and FMI

For comparing the two approaches of FDMU and FMI, the following aspects have to be considered:

- Type of coupling
- Coupling technology
- Programming language bindings
- Co-simulation algorithms
- Implementation

Further aspects may be interesting for certain user communities and will be investigated in the near future. The main focus of this paper is to figure out advantages of both technologies and to make a first proposal for a good combination of them.

There is no doubt that each of the co-simulation frameworks will find their friends in many fields of applications such as Automotive and MEMS* design. But there will be situations, too, where a combination of the power of FMI and the flexibility of FDMU will be the most appropriate solution.

4.1 Type of coupling

FMI for Co-Simulation defines a tight coupling between the FMI master and a co-simulation component denoted as FMU acting as slave (*Figure 2*). The interface specification contains a set of C function prototypes. This means all couplings are based on

local function calls. The FMI master is the caller, all FMU slaves act as callee. Unfortunately, this clean architecture is slightly broken by a few callback functions e.g. for memory management and notifications.

FDMU is based on a loose coupling between the FDMU Console, the FDMU Master and all FDMU Wrappers (*Figure 1*). The FDMU framework is implemented as a service-oriented architecture (SOA). Each component excepting the FDMU Console (the user's front-end) works as service within a client-server infrastructure. The FDMU API is based on the Web Service standards (WS-I, WS-Attachments, WS-Security, WS-Notification, ...) [11].

4.2 Coupling technology

The main focus of the FMI specification is on efficiency. Using function calls within one process and one memory domain means minimal communication overhead. Currently no FMI benchmarks can be found in the literature. But the authors of this paper guess that the highest performance can be achieved by using the FMI approach for co-simulation in contrast to SOA-based approaches.

The FDMU framework uses HTTP-based SOAP messages for signal data exchange between the co-simulation components as required by the Web Services standards. This approach allows very flexible couplings between different hosts, hardware platforms, operating systems, and IP domains. The main focus of the FDMU approach is on flexibility. FDMU users can couple simulators between different departments or companies via Internet. The Web Services provide secure, encrypted, and standardized communication through firewalls. The drawback of this flexibility is the communication overhead.

Furthermore, the software architecture of the FDMU framework is based on the paradigm of distributed systems. Concurrency and multi-threaded implementations are supported and all communication methods include thread-safe queues for distributed and deadlock-free transmission of data. This architecture ensures scalability for large-scale co-simulation scenarios with more than four or five FBB.

Currently the FMI specification for co-simulation does not contain any details on multi-threaded implementations. Otherwise the specification explicitly supports asynchronous execution of API functions (e.g. `fmiDoStep()`) which means parallel simulation is allowed for FMI slaves and FMI is prepared for concurrency. The reference implementation which will be prepared within the MODELISAR project by the Fraunhofer IIS/EAS is based on a single-threaded approach. The FMI Master controls all

* microelectromechanical systems

FMI Slaves using a pure sequential algorithm. More details concerning the asynchronous features and multi-thread aspects will be investigated in further project research.

4.3 Programming language bindings

The FMI specification defines a set of C API functions. For the coupling between the software components (FMI master, FMI slaves), dynamic link libraries (DLL on Windows) or shared objects/libraries (SO on Linux/UNIX) are used. If all the needed DLLs (or SOs) are available, this approach is very simple and user-friendly. But the tight coupling between the different components works properly if binary and/or platform compatibility (32/64 bit, x86, x64, sparc, powerpc) are fulfilled and function calls are used for communication. If two FMUs run in different processes, operating systems or machines, a communication layer is needed. Currently this layer is not prescribed by the FMI specification.

For the FDMU framework, no language binding is required. The coupling interface is completely defined by the WSDL (Web Service Definition Language) specification of the FDMU Master. Each provider of an FDMU Wrapper for a certain simulation tool can use this WSDL specification to generate the code fragments for their own implementation for the needed programming or scripting language like Java, C, C++, C#, Python or Perl. All Web Service frameworks like JAX-WS, .Net WCF, AXIS, CXF, or similar can be used if they are compatible to the W3C standards. Unfortunately, there are some pitfalls and interoperability problems when using the different Web Service frameworks. So the current implementation of the FDMU framework is mostly based on Java

and JAX-WS with a small adapter layer for tool-specific components for C, C++, and C#.

4.4 Co-simulation algorithms

The FMI specification for co-simulation defines a sequential computation flow for the master algorithm. The master analyzes the connection graph between all co-simulation components (slaves) and calculates an appropriate invocation order for the signal update and simulating the next time step of all components/simulators. Furthermore, the FMI Master can control in a very fine-grained way the simulation progress of each coupled simulator if the simulator provides the corresponding capabilities. Using these facilities it is possible to provide co-simulation algorithms with step size control or iterations at certain time instants.

The primary focus of the FDMU Master is to provide the concurrent signal flow between all FBB. The simplest FDMU Master algorithm handles all connections independent from each other. This FDMU communication schema prevents dead-locks and allows maximizing the throughput at least for signal connections which are independent from each other. Furthermore, the FDMU approach allows other algorithms e.g. with adapting the simulation step size during a running simulation. But the functionality is more limited here with respect to step size control of FMI.

4.5 Implementation

For FMI the provider of the simulation software has to implement the FMI interface as described in the specification. In this case the main advantage for the user is that he can use FMI-enabled tools without any further investments. For simulation tools without

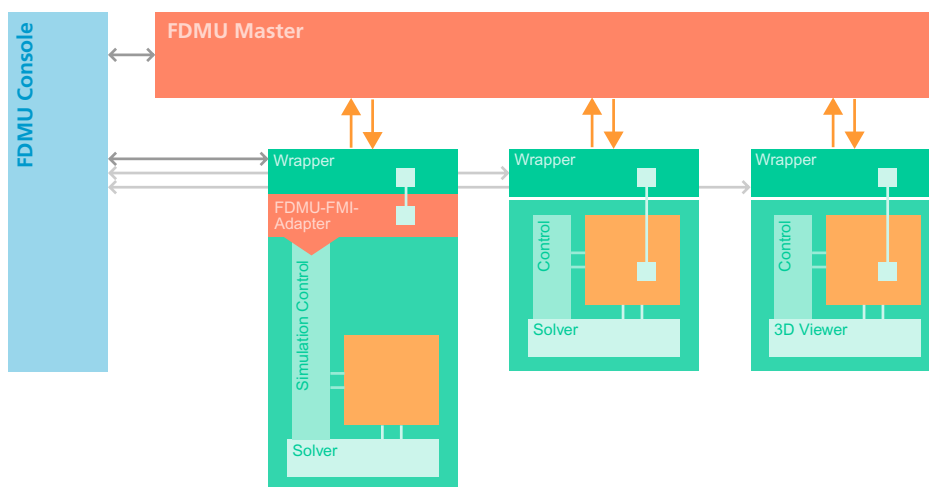


Figure 3: Idea of integrating FMI under FDMU

any FMI support, an FMI-based co-simulation is not possible.

FDMU has another strategy here. A FDMU Wrapper encapsulates a simulation tool using a small software layer around the tool. The advantage is that the simulator remains untouched and open interfaces (external function interface, tool-specific API, file IO ...) are used only. The disadvantage is there is needed extra wrapper software. But for simulators without any FMI support this approach could be the only solution for using it within a co-simulation framework.

4.6 Summary

Both approaches, FMI and FDMU, have their benefits and problems. At a first glance FMI and FDMU represent two different perspectives to co-simulation-based system design:

- FMI is focused on efficient co-simulation interfaces between electrical, mechanical, and software models. The primary goal is to simulate and analyze the models.
- FDMU addresses the combination of both the behavioral models and the 3D geometry (CAD) data into one simulation approach. The primary goal is to provide an interactive 3D visualization with functional simulation in background.

Despite this there are similarities too:

- All mathematical aspects of co-simulation, like numerical error introduction by tearing, interpolation at the interface, the need of suitable, intelligent Master algorithms are similar in both approaches.
- The slave simulators have to be prepared for participating in the coupled simulation.

5 Proposals for combining FDMU and FMI approaches

In this chapter we propose three options for a combination of FDMU and FMI components. A complete integration of the two frameworks into one solution may be useful in the future. Considering the differences in the underlying software architecture, a simpler solution using adapters between FMI and FDMU seems more appropriate in the near future.

In general, there are two options for the implementation of adapters. For FDMU users, FMI slave components can be integrated in the FDMU framework via an FDMU wrapper (*Figure 3*). This wrapper completely encapsulates an FMI component. The wrapper has to emulate all the needed FMI Master function calls and call-back functions.

For FMI users, an adapter between FMI Master and FDMU Wrapper is needed. The adapter can be implemented by an FMI Slave component, which provides the Web service interface to an FDMU Wrapper (*Figure 4*).

The advantage of both solutions is that a user continues to work with his or her favourite framework and has access to additional components of the other framework. The disadvantage of the adapter-based solution is that only a subset of framework functions can be supported. For instance, it is not possible to emulate FMI call-back functions within a Web Service-based framework like FDMU in an efficient way. Further investigation of the details of the adapter-based solutions is needed.

Last but not least a third approach seems reasonable: Both frameworks can be coupled via a bridge (*Figure 5*). In this case a bidirectional communication between FMI Master and FDMU Master is needed. In contrast to the adapter-based solutions this approach is not based on open interfaces. It means a

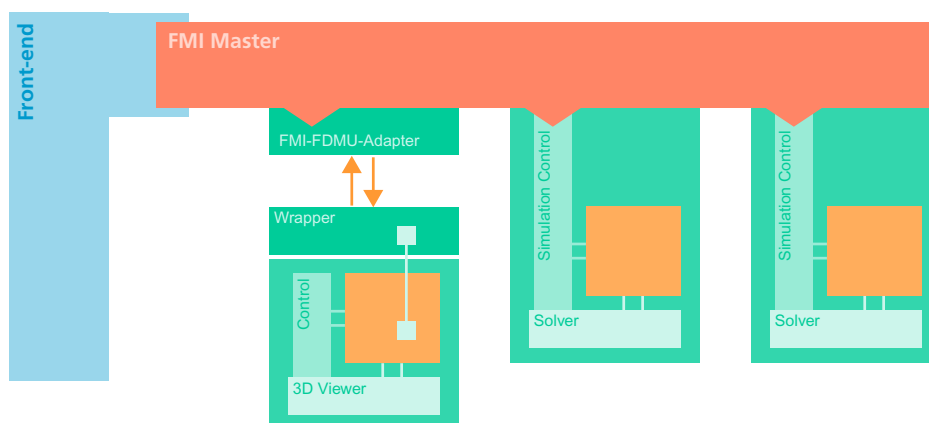


Figure 4: Idea of integrating FDMU under FMI

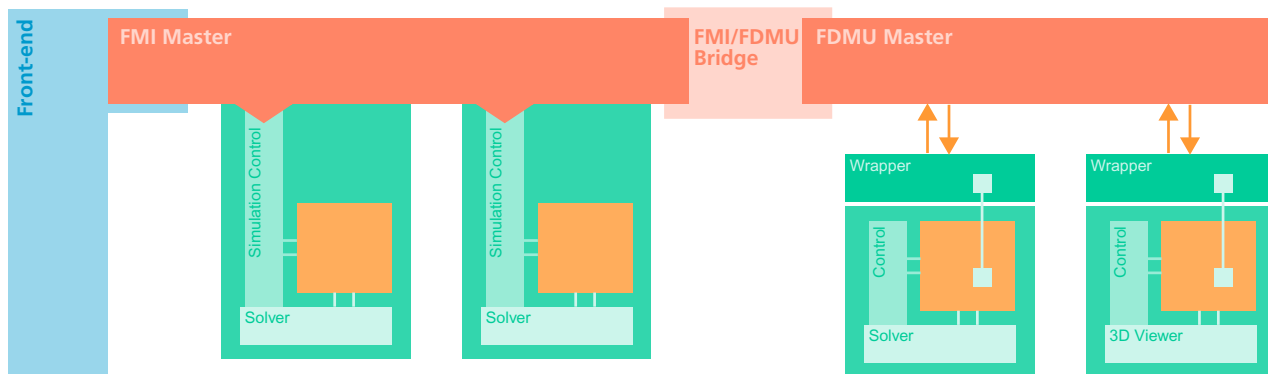


Figure 5: Idea of bridging FMI and FDMU

bridge implementation needs access to the internal interfaces of the two masters. As the authors of this paper are involved in the development of both FMI and FDMU this restriction is less important. Finally, the bridge solution could be a first step forward in unifying FMI approach as well as FDMU approach in the future.

6 Summary

The paper compares the two simulation approaches for heterogeneous systems called *Functional Digital Mock-up* (FDMU) and *Functional Mock-up Interface* (FMI). Both principles are well suited for a comprehensive investigation of multi physical systems. The main ideas of both approaches are shortly presented.

The FDMU approach focuses on a web service-based architecture for a flexible combination of different simulators and the involvement of an interactive visualization. Within this approach, the simulation tools can be used without additional implementations. However, the models must use one-directional connectors which may cause slight modifications.

The main ideas of the FMI specification are characterized by model exchange as well as co-simulation. The first idea realizes a passing over of sub-models between different simulation tools. The second way defines a tight coupling between a so-called master simulation tool – acting mainly as caller – and all other simulation tools which are acting as slaves. For establishing the FMI ideas, the vendors have to implement additional code into their tools.

Both approaches have advantages and disadvantages. However, a combination of both principles seems to be possible and promising. Three opportunities for such a combination are shortly presented in

the paper. All three ideas ought to be proved more particularly in the future.

References

- [1] Bastian, J.; Clauß, C.; Wolf, S.; Schneider, P.: Master for Co-Simulation Using FMI. 8th International Modelica Conference, Dresden, Germany, March 20-22, 2011.
- [2] Enge-Rosenblatt, O.; Schneider, P.; Clauß, C.; Schneider, A.: Functional Digital Mock-up – Coupling of Advanced Visualization and Functional Simulation for Mechatronic System Design. Proc. ASIM Workshop, Ulm, March 4-5, 2010.
- [3] Modelisar: <http://www.modelisar.org>
- [4] Noll, C.; Blochwitz, T.; Neidhold, T.; Kehler, C.: Implementation of Modelisar Functional Mock-up Interfaces in SimulationX. 8th International Modelica Conference, Dresden, Germany, March 20-22, 2011.
- [5] Otter, M.; Blochwitz, T.; Elmquist, H.; Junghans, A., Mauss, J., Olsson, H.: Das Functional-Mockup-Interface zum Austausch Dynamischer Modelle. Keynote at ASIM Workshop, Ulm, March 4-5, 2010.
- [6] Relovsky, B.: Overview of ITEA2 Project MODELISAR (I). Keynote at the 7th International Modelica Conference, Como, Italy, September 20-22, 2009.
- [7] Schneider, P.; Clauß, C.; Enge-Rosenblatt, O.; Schneider, A.; Bruder, T.; Schäfer, C.; Voigt, L.; Stork, A.; Farkas, T.: Functional Digital Mock-up – More Insight to Complex Multi-physical Systems. Multiphysics Simulation – Advanced Methods for Industrial Engineering, 1st International Conference, Bonn, Germany, June 22-23, 2010, Proceedings.
- [8] Schubert, C.; Thomas Neidhold, Guenter Kunze: Experiences with the new FMI Standard - Selected Applications at Dresden University. 8th International Modelica Conference, Dresden, Germany, March 20-22, 2011.
- [9] Stork, A.: FunctionalDMU – Eine Initiative der Fraunhofer Gesellschaft, 2006. <http://www.functionaldmu.org>
- [10] Stork, A.; Thole, C.-A.; Klimenko, S.; Nikitin, I.; Nikitina, L.; Astakhov, Y.: Simulated Reality in Automotive Design. International Conference on Cyberworlds, Hannover, 2007.
- [11] W3C: Web Service Standards WS-*, 2010. <http://www.w3.org/standards/webofservices/>

Towards Design Optimization with OpenModelica Emphasizing Parameter Optimization with Genetic Algorithms

Hubert Thieriot^a, Maroun Nemer^a, Mohsen Torabzadeh-Tari^b, Peter Fritzson^b, Rajiv Singh^c, John John Kocherry^c

^aCenter For Energy and Processes, MINES ParisTech, Palaiseau, France

^bPELAB Programming Environment Lab, Dept. Computer Science, Linköping University, SE-581 83, Sweden

^cEvonik Energy Services, Pvt. Ltd., Corporate Office, Noida 201 301, India

Abstract

One of the main goals when modeling a physical system is to optimize its design or configuration. Currently existing platforms are often dependent on commercial software or are based on in-house and special-purpose development tools. These two alternatives present disadvantages that limit sharing and reusability. The same assessment has partly motivated the origin of the Modelica language itself. In this paper, a new optimization platform called OMOptim is presented. Intrinsically linked with OpenModelica, this platform is mainly aimed at facilitating optimization algorithm development, as well as application use together with models. A first version is already available and three test cases of which one using respectively Dymola and two using OpenModelica are presented. Future developments and design considerations of OMOptim but also of related OpenModelica computation functions are also discussed.

Keywords: Optimization, model-based, parameter, genetic algorithm, Modelica, modeling, simulation

1. Introduction

Model-based product development is an approach where a computer-based model of the product is built and refined before the actual production, to reduce costs, increase quality, and shorten time-to-market. Optimization is often used to improve product quality or design. Several types of optimizations can be used with these goals in mind. This can either concerns parameter or configuration optimization (e.g. which selection of the best components or connection paths to use in a defined process). Some design tasks also need a dynamic optimization to benchmark different configurations. For the user but also for the developer of such algorithms, two main issues can be noticed. The first issue concerns the development platform itself. The developers can either use a commercial platform (e.g. MatLab connected

with a external simulator) or develop their own environment. The disadvantages of the first option are mainly the proprietary aspects of such tools which makes it harder to modify and extend, and also the involved license fees. The latter solution needs more development time and reduce exchange opportunity with other teams. Another important issue of model-based optimization lies in the computation time. Optimization applications often requires a large number of iterations and thus, a long time to give interesting results. This paper presents an initiative to limit these two main issues by developing an open-source optimization platform for OpenModelica (OMOptim) involving generation of efficient source code for multi-core computer architectures for increasing simulation performance.

1.1. Structure of the Paper

This paper first presents the context and motivation of the OMOptim development. A general review of optimization methods is then presented. The next sections successively describe the first version of OMOptim, an example of an application already

Email addresses: hubert.thieriot@mines-paristech.fr (Hubert Thieriot), maroun.nemer@mines-paristech.fr (Maroun Nemer), mohsen.torabzadeh-tari@liu.se (Mohsen Torabzadeh-Tari), peter.fritzson@liu.se (Peter Fritzson), r.singh@evonik-es.in (Rajiv Singh), jj.kocherry@evonik-es.in (John John Kocherry)

implemented and some concluding words about the intended future of this platform.

2. Requirements

The Center for Energy and Processes of Mines ParisTech school is involved in the CERES project [1] concerning industrial processes optimization. In this project, the best process technologies and heat recovery topology should be chosen simultaneously with mini-mum costs and environmental impacts. PELAB on the other hand is involved in the SSF Proviking EDOp project [2], concerning dynamic optimization for large industrial optimization problems, targeting both para-metric as well as dynamic optimization. This paper aims at building a bridge between these two projects with a common open source optimization platform. Thus, it should be ergonomic and efficient enough to use but also allow development of algorithms in the environment. A first version of this tool called OMOp-tim has been developed and is described below. Besides this goal, one critical issue will be the simulation (and thus) optimization time. Therefore, optimization algorithms but also the simulation tool efficiency should be very high. This paper briefly presents current and intended developments which go in this direction.

3. Optimization

This project aims at solving several different optimization problems, and in order to do this efficiently, a number of different solution techniques are required. Optimization problems can be classified according to several criteria e.g. existence of constraints, the nature of variables, and the nature of equations involved. A large number of optimization algorithms have been developed over the last decades to solve these different problems. One can roughly divide them in two families: gradient based methods and meta-heuristics algorithms.

3.1. Gradient based methods

The gradient based family contains numerical linear and non linear programming methods. These algorithms require substantial gradient information and are often used to improve a solution near a starting point. Applied on simple models, they offer an efficient way to find global optimum. However, many

engineering optimization problems are highly non-linear and present several optima. Such problems create numerical difficulties (like discontinuities) for this family algorithms and result can depend on initial point defined by the user.

3.2. Meta-heuristic algorithms

Meta-heuristic algorithms present a common characteristic: they combine rules and randomness to imitate natural phenomena. Within such methods, derivative computation is unnecessary. Most developed methods are evolutionary algorithms and genetic algorithms which are based on biological evolution formulation [3] but also tabu search, which reproduces animal behavior [4]. Simulated annealing is another meta-heuristic method based on physical annealing process [5].

3.2.1. Genetic algorithms and evolution strategies

A genetic algorithm (GA) is based on natural evolution and reproduces its main operations: reproduction, crossover and mutation. The initial theory has been proposed by Holland [6] and Goldberg [3] among others. An individual is represented by a genome which contains values of decisive parameters. For each individual, fitness values are calculated; these fitness values correspond to the objectives we want to minimize or maximize. A population is initially created by assigning random values to decisive parameters for each individual. New generations are created by combination of parents and innovation is introduced by mutation step. At each generation, a selection operation is followed which keep only the best individuals according to the fixed objectives but also following diversity parameters. Evolution strategies mainly differ from Genetic algorithms (GAs) in parameters coding: while GAs use binary coding and operations, evolution strategies use real coded parameters [7]. By extension, evolution strategies are often called genetic algorithms. These methods have largely been applied to estimate parameter values which minimize one or several objectives. It is indeed independent of problem type and can be applied to constrained or unconstrained problems, can have discrete or continuous variables, can follow one or several objectives and can be applied to linear or non-linear problems. Evolution strategies and more generally meta-heuristic algorithms present several advantages. First, they can be applied to complex engineering problems. They also do not need any

particular initialization point and are therefore independent of it. Finally, they tend to escape local optimum problems (e.g. with highly discontinuous problems). However, for linear and simple non-linear problems, linear or non-linear programming methods are much more suited and efficient (especially because of specific formulation and gradient information).

4. OMOptim 0.9

4.1. Goals

OMOptim intends to be a platform where different families of optimization algorithms can be implemented and linked with the OpenModelica simulator but also with other tools e.g. using FMI (Functional Mock-up Interface) [8]. Figure 1 illustrates its high-level design concept.

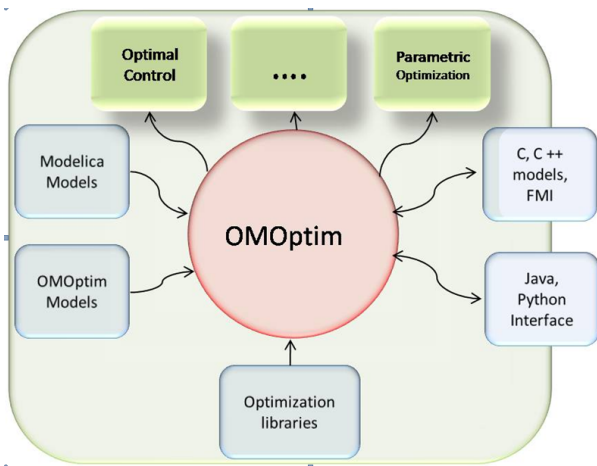


Figure 1: Top-level conceptual view of the OMOptim model-based optimization tool in OpenModelica.

4.2. Implementation

A first version of OMOptim including a graphical user interface has been developed in C++ and already tested on several use-cases (cf. Section 5). This version uses the OpenModelica API to read and eventually modify the model through the Corba communication protocol [9].

This version can only run meta-heuristic optimization methods since at this time, it does not have access to information about derivatives, even though OpenModelica can produce such information. As previously stated, only input variables specification and output variables reading are needed for such methods. Specifying input variables and reading results

is done using input and output text files. To implement meta-heuristic algorithms, an efficient and adapted framework has been used (ParadisEO library [10]). OMOptim already includes several genetic algorithms, e.g. NSGA2, SPEA2 [11] or self-adaptive versions [12].

4.3. User interface

At the same time, a GUI has been developed allowing graphical selection of optimization variables, parameters and objectives (Figure 2) but also reading results.

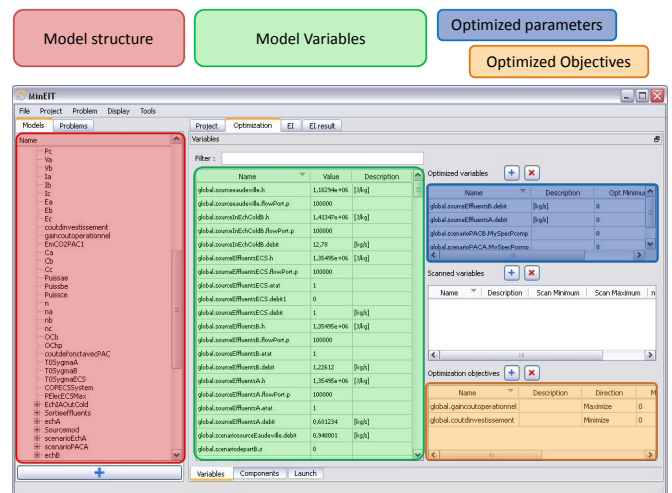


Figure 2: Parameters and objectives selection in the OMOptim optimization problem definition.

5. Test cases

Three test cases are presented here. The first uses Dymola as a simulation tool on an industrial application, but still uses OpenModelica to access the model structure. The second shows a small example application with OpenModelica. The third uses OpenModelica on an industrial application and an optimization module which is currently executed separately, but will be integrated with OpenModelica. As previously stated, meta-heuristic algorithms can interact with simulation tool using only input and output files. Thus, it is possible to interact with most simulation tools. However, in the future, all OMOptim algorithms may not be compatible with other simulators than OpenModelica (cf. section 6)

5.1. Heat-pump application using Dymola for simulation

A first application has been done which concerns a multi heat-pump system in a food industrial process [13]. This system consists of three heat-pumps used to heat-up solutions of the process. These three heat-pumps are connected to a heat-recovery stream. The model integrates dynamic items e.g. hot water tank emptying and filling during simulation (cf. Figure 3).

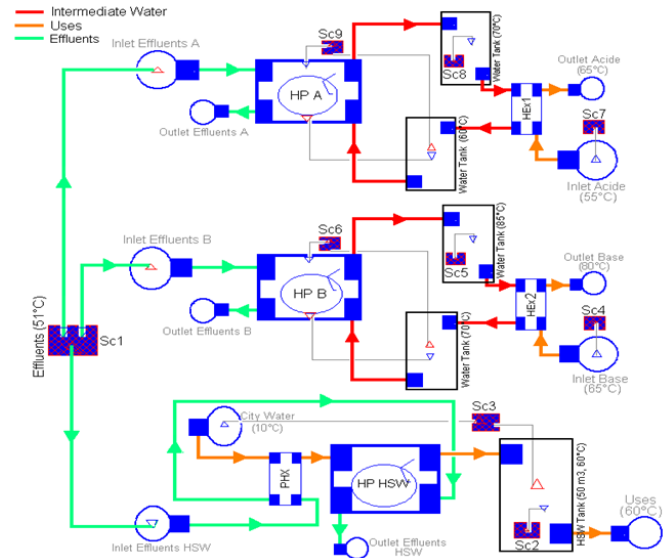


Figure 3: Modelica model of an industrial process being optimized.

The optimization consists in finding optimal flow repartitioning of the heat-recovery stream but also optimal powers of these heat-pumps, including the possibility to disable one or several heat-pumps. Two objectives are considered in this optimization: decreasing operational cost and investment cost. An auto-adaptive genetic algorithm has been developed for this study in OMOptim [13][12]. This genetic algorithm includes standard deviation of each genome parameter in the genome itself of the genetic algorithm. Therefore, the variation amplitude between each generation is itself submitted to modification and selection.

OMOptim allows the user to obtain several optimal configurations according to the two objectives followed i.e. investment and operating cost. Moreover, a sensitivity analysis has been performed to analyze the impact of CO2 carbon tax on optimum configurations (Figure 4).

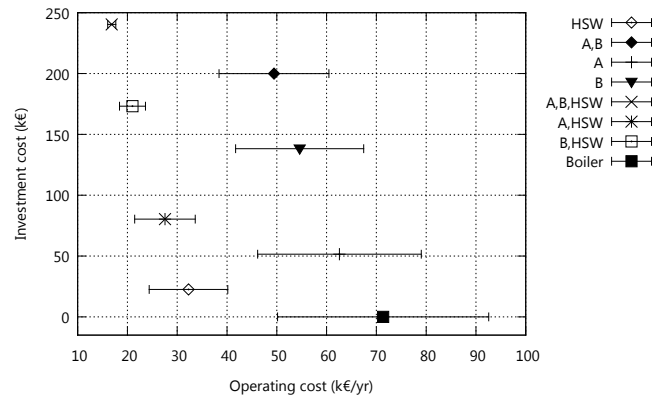


Figure 4: Investment and operation costs for optimal configurations (horizontal bars correspond to carbon tax variation sensitivity).

5.2. A Linear actuator application using OpenModelica

The model here consists of a linear actuator with a spring damped stopping [14, p. 583]. The model configuration is presented on Figure 5.

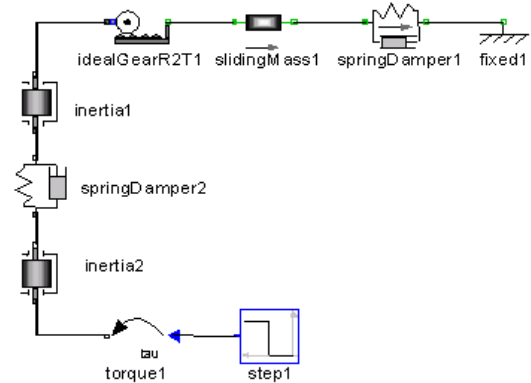


Figure 5: Linear actuator model

A reference response is generated considering a first order system. This response is defined by a first order ODE : $0.2 * \dot{y}_{ref}(t) + y_{ref}(t) = 0.05$. The optimization consists in making the resulting linear actuator behavior be as close as possible to this reference response. To achieve this, the damping parameters d_1 and d_2 of both spring dampers are considered as free variables to be determined by the optimization algorithm. The objective function corresponds to the integral of square deviation along simulation time T : $f(d) = \int_0^T (y(t) - y_{ref}(t))^2 dt$.

With obtained parameters ($d_1 = 4.90$ and $d_2 =$

19.88), the behavior suits the reference response well (cf. Figure 6). These results were obtained in less than five minutes on a standard Intel Core2 Duo @ 2.53 GHz.

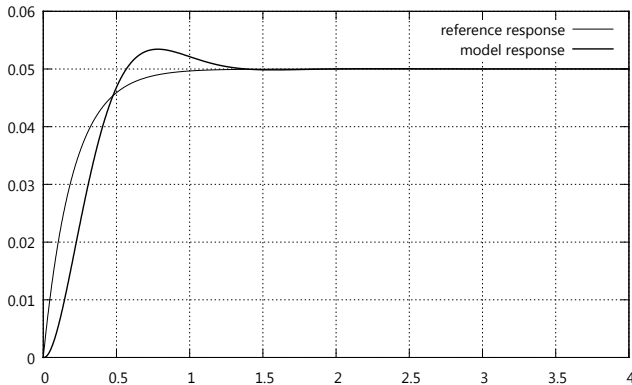


Figure 6: Model and reference responses after optimization of damper parameters

5.3. A dynamic optimization using an external SQP module

This third test case concerns a power plant regulation. It is only described very briefly here - a more detailed presentation is planned in a future paper. This application has been run using an external Sequential Quadratic Programming (SQP) optimization module.

In power plants, the main steam temperature control regulates the spray (attemperator) flow rate. Precise modeling of super heater dynamics and improving the quality of control of the superheated steam temperature is essential to improve the efficiency of the Boiler. In addition to this, the physical constraints of the turbine blades are also met using this control strategy. This control methodology is based on an adaptive prediction of the steam temperature trends. The architecture of the newly developed control system is similar to that of conventional boiler but the temperature feedback is given from the model instead of a sensor as shown in Figure 7.

A simple heat exchanger model is adapted to model the first stage of super heater regarding steam temperature, steam flow and flue gas temperature as measurements. This resulted into a set of algebraic differential equations which captured the behavior of the super heater along with the attemperator.

A SQP optimizer is used to calculate the spray flow, driven by an objective function to find the least

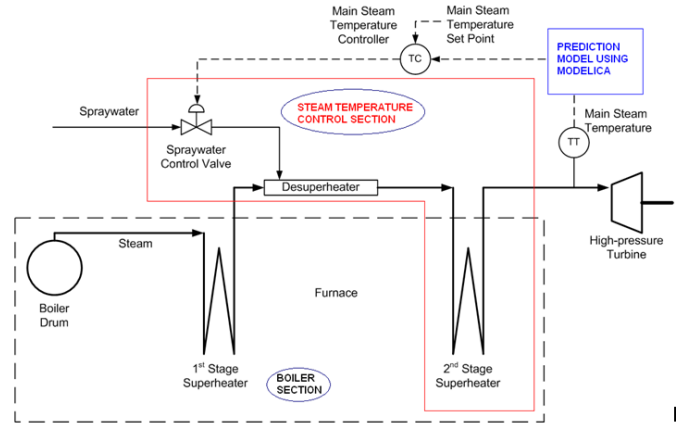


Figure 7: Advanced steam temperature control strategy for power plant

square error between the predicted and set point of steam temperature for a defined control horizon. Dynamic constraints are considered for spray and metal temperatures to consider the metal strains.

The first results are promising. However, this function is a separate module that is not yet integrated with the available version of OMOptim. This integration is planned in the near future.

6. Future work

6.1. OMOptim Structure Evolution

OMOptim intends to become an attractive framework to develop and execute optimization algorithms for Modelica users. To achieve this, its structure should be flexible enough to address the needs of many different kinds of optimization. The structure should also provide an efficient and ergonomic way to develop special-purpose algorithms including sharing and usage. Like a Modelica library, it would be pertinent and useful to list available optimization algorithms in libraries sharable within the Modelica community. Moreover, the structure should be able to support the combination of several algorithms working together. It should for example be used to apply a meta-heuristic optimization function with an objective function computed from another function (e.g. the objective could itself be the result of a sensitivity analysis). In some cases, it should also be possible to create new algorithms by graphically connecting existing optimization modules like in component-based modeling.

6.2. Hybrid Optimization

Meta-heuristic optimization algorithms can be coupled with local search functions [15]. This combination intends to combine advantages of both families. Meta-heuristics allow spreading populations over a large domain and thus limit the risk of obtaining a local optimum solution. Local search functions can lead to a faster convergence and to more precise results (e.g. [16] or [17]).

To achieve hybrid optimization implementation, a stronger link with OpenModelica should be built. In particular, gradient information should be communicated to optimization methods. The first developments in this direction are currently under way.

6.3. Dynamic optimization

Dynamic optimization requires modifying model parameters while performing the simulation. This functionality assumes the development an interface between OpenModelica and OMOptim while the former is computing. First trials have been done in this direction, using the new online interactive simulation facility of OpenModelica. More specifically, integration of a Sequential Quadratic Programming optimizer within OMOptim is planned in the near future (cf. section 5.3).

6.4. Parallelization for Efficient Computation

Applying parallelism and parallel compilation techniques at many levels of the problem, from problem formulation to inlining the solver and software pipelining, is being addressed in this project [18]. The constraints of the optimization problem can often be handled in parallel. In this case large system models can be restructured to smaller sub-system models. The PELAB research group at Linköping University has a long tradition of handling the compilation process in parallel, optimizing it, and adapting it for multi-core architectures. Some recent encouraging results [19] about using GPU architectures instead of CPU caused PELAB to invest in a two-teraflop (peak) Nvidia Fermi GPU that will be used in this project. Another step is to extend the support of efficient event-handling in parallelized code in order to also handle hybrid models.

6.5. Optimization Performance Profiling and Debugging

One current disadvantage of using high-level equation based languages [14] as well as other high-level

simulation tools is the poor support for performance profiling and debugging. This will be even more pronounced when an engineer wants to trace the reason to why an optimization is too slow or has failed. There exists a substantial expertise at PELAB regarding debugging and traceability technology in integrated environments. We are planning to use this as a basis for a profiling feature in the optimization platform that is needed for tracing the causes of problems bottle-necks in the model.

7. Related Work

7.1. jModelica

The current Modelica language does not include formulating optimizations problems. However, a language extension called Optimica [20] has been developed by JModelica (www.jmodelica.org). JModelica offers an efficient platform for dynamic optimization and works in close collaboration with the model since it has an integrated Modelica compiler.

7.2. Dymola optimization library

The Dymola commercial tool from Dassault Systems [21], Dymola has its own optimization library, containing genetic algorithms. Another product from Dassault Systems is Isight [22] that supports process flow optimization with genetic algorithms. The main disadvantage of these two products is their closeness.

7.3. Meta-heuristic algorithms

Several tools may link meta-heuristic optimization methods to different simulators. One can cite OptiY [23], modeFrontier [24], Isight [22], or GenOpt [25]. They propose a rich list of implemented algorithms and can be used with nearly all simulators (all these tools interact with simulation software using input file modification and output file reading). Excepting GenOpt, all these softwares are commercial.

7.4. What should OMOptim offer

OMOptim aims to offer OpenModelica users an extension opening new opportunities. Especially, it intends to be a shared and open platform where scientists could develop optimization algorithms and apply them to Modelica models.

OpenModelica has been chosen for its opening and its substantial development rhythm. Also, OpenModelica supports symbolic differentiation which allows robust and advanced numerical methods, very

useful in optimization problems. This could be especially useful for the development of hybrid algorithms (cf. section 6.2).

Parallelism is also an intended development direction. Applying parallelism and parallel compilation techniques at many levels of the problem, from heuristic simulation repartition to inlining the solver and software pipelining, is being addressed in this project. For example, population based meta-heuristic optimization methods present high parallel scalability.

Concerning dynamic optimization or components/connections that change during simulations, the Modelica language doesn't yet support structural dynamism, i.e. changes in the causality during simulations. However, with a little relaxation of this requirement the environments would be much flexible and better suited for optimization tasks [26].

References

- [1] Ceres project : www.eceer.com/eceer_i_15/.
- [2] Edop project : <http://openmodelica.org/index.php/research/omoptim/edop>.
- [3] D. Goldberg, Genetic algorithms in search, optimization, and machine learning, Addison-wesley, 1989.
- [4] F. Glover, Future paths for integer programming and links to artificial intelligence, *Comp. Oper. Res.* 13 (1986) 533–549.
- [5] S. Kirkpatrick, C. Gelatt, M. Vecchi, Optimization by simulated annealing, *Science* 220 (1983).
- [6] J. Holland, *Adaptation in Natural and Artificial Systems*, Ann Arbor, MI, 1975.
- [7] T. Bck, H. Schwefel, An overview of evolutionary algorithms for parameter optimization, *Evolutionary computation* 1 (1993) 1–23.
- [8] Itea2, modelisar : www.itea2.org.
- [9] Openmodelica system documentation, available on <http://www.openmodelica.org>.
- [10] A. Liefooghe, M. Basseur, L. Jourdan, E.-G. Talbi, *Paradiseo-moeo: A framework for evolutionary multi-objective optimization*, 2007.
- [11] E. Zitzler, M. Laumanns, L. Thiele, Spea2: Improving the strength pareto evolutionary algorithm, in: *EUROGEN*, Citeseer, 2001, pp. 95–100.
- [12] K. Deb, H. Beyer, Self-adaptation in real-parameter genetic algorithms with simulated binary crossover, in: *Proceedings of the Genetic and Evolutionary Computation Conference*, Citeseer, 1999, pp. 172–9.
- [13] R. Murr, H. Thieriot, A. Zoughaib, D. Clodic, Multi-objective optimization of a multi water-to-water heat pump system using evolutionary algorithm, Submitted to *Applied Energy*. ()
- [14] P. Fritzson, *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*, Wiley-IEEE Press, 2004.
- [15] A. Hopgood, L. Nolle, A. Battersby, Hybrid genetic algorithms: A review (2006) –.
- [16] M. Fesanghary, M. Mahdavi, M. Minary-Jolandan, Y. Alizadeh, Hybridizing harmony search algorithm with sequential quadratic programming for engineering optimization problems, *Computer methods in applied mechanics and engineering* 197 (2008) 3080–3091.
- [17] S. Katare, A. Bhan, J. Caruthers, W. Delgass, V. Venkatasubramanian, A hybrid genetic algorithm for efficient parameter estimation of large kinetic models, *Computers & chemical engineering* 28 (2004) 2569–2581.
- [18] H. Lundvall, P. Fritzson, Automatic parallelization using pipelining for equation-based simulation languages, in: *In proceedings of the 14th Workshop on Compilers for Parallel Computing (CPC'2009)*, Zurich, Switzerland.
- [19] P. stlund, K. Stavker, P. Fritzson, Parallel simulation of equation-based models on cuda-enabled gpus, Submitted to *EuroPar* (2010).
- [20] J. Akesson, K.-E. Arzen, M. Gafvert, T. Bergdahl, H. Tummescheit, Modeling and optimization with optimica and jmodelica.org–languages and tools for solving large-scale dynamic optimization problems, *Computers & Chemical Engineering* 34 (2010) 1737–1749.
- [21] Dassault systemes, www.3ds.com.
- [22] Isight product page : <http://www.simulia.com/products/isight.html>.
- [23] Optiy, multidisciplinary analysis and optimization, <http://www.optiy.eu>.
- [24] Modefrontier, <http://www.modefrontier.com>.
- [25] Genopt, <http://simulationresearch.lbl.gov/go/>.
- [26] H. Nilsson, G. Giorgidze, Exploiting structural dynamism in functional hybrid modeling for simulation of ideal diodes, in: *Eurosim 2010*.

From Physical Modeling to Real-Time Simulation : Feed back on the use of Modelica in the engine control development toolchain

Zakia Benjelloun-Touimi¹; Mongi Ben Gaid¹; Julien Bohbot¹; Alain Dutoya²;
Hassan Hadj-Amor¹; Philippe Moulin¹; Housseem Saafi³; Nicolas Pernet¹

¹IFP Energies nouvelles
1 & 4, avenue de Bois-Préau, 92852 Rueil-Malmaison Cedex, France

²INCKA SA
85 avenue Pierre Grenier, 92100 BOULOGNE-BILLANCOURT

³Ecole Nationale d'Ingénieurs de Sousse
Technopôle de Sousse, 4054 Sousse, Tunisie

Abstract

This article provides feedback from using Modelica in the "System Modelling" area, involving modelling (behavioural and dynamic modelling), direct simulations, control and real-time applications.

The described work was undertaken within three European projects: Eurosyslib, Modelisar and Open Prod.

Our aims are to attest Modelica language in an overall model of a vehicle consisting of vehicle dynamics, combustion engine, transmission, drive line, brakes and control systems.

ModEngine is a complete IFPEN¹ library, resulting from our participation in those European projects. It allows the modelling of a complete engine with diesel and gasoline combustion models. It may be interfaced with control algorithms written in Simulink thanks to the new Functional Mock-up Interface specification from Modelisar project.

Both versions under commercial software Dymola and free one OpenModelica are available.

Feedback will concerns also problems encountered and advantages in use Dymola and OpenModelica platforms.

Keywords: Control, Eurosyslib, FMI, Library, Modelica; Modelisar; ModEngine, Modelisation, Openprod, Simulation, Real-time.

1 Introduction

The use of the ModelicaTM language [19] for hierarchical physical systems modelling is booming thanks to an international effort, but mainly because it meets what engineers and researcher expect for their development today.

In this paper we propose to provide a feedback about our experience in using Modelica for an industrial application in European projects (Eurosyslib [23], Modelisar [24], and OpenProd [25]): the development of control strategies for automotive engines. In this field, IFPEN [22] has promoted an approach where simulation tools play a crucial role at the different stages of the development process. This approach has been described in various publications. It utilizes the modelling library IFP-Engine developed in C language, commercially available under the LMS Imagine.Lab AMESim [26] environment.

The different stages of the development process and the associated needs for simulation are the following:

1. System understanding: a detailed model of the system is needed, including the representation of all the physical phenomena. The physical accuracy is important.
2. Control strategy development: the focus is on the control part, the system model is not modified. However, it is coupled with the control strategies, executed in Simulink. The execution platform is important.

¹ IFPEN : IFP Energies nouvelles, new name of IFP

3. Control validation: the model must be compiled and executed in an environment that is representative of real-time. The ability to compile and export the model is important.
4. System integration: the interactions between the engine and the other parts of the vehicle (transmission, after treatment,) are considered, the engine model and its control must be coupled to other component models, developed in various modelling environments. The model integration capabilities are important.

The plan of this paper will follow the structure given by these different stages. Each of them will be described in detail in a different section that details the design choices, the advantages and drawbacks of Modelica in this context.

2 System understanding: development of ModEngine library

Requirements for the ModEngine library were derived from the existing IFPEN AMESim library (Engine). The users can quickly assemble blocks that result in vehicle simulators. ModEngine is now functional in Dymola [20] and OpenModelica [21]; it contains more than 250 sub models. We continue to contribute and to optimize it in order to obtain accurate and fast calculations for control and real-time applications, respecting the procedure described by figure 1 below.

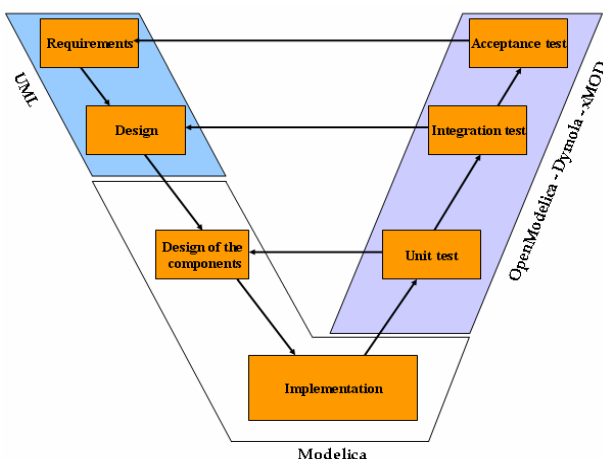


Figure 1: Modelling/Simulation/Control/Real-time validation Cycle

The ModEngine library has been developed to allow the simulation of a complete virtual engine using a characteristic time-scale of the order of the crank-

shaft angle. A variety of elements are available to build representative models for engine components, such as turbocharger, wastegate, gasoline or Diesel injectors, valve, air path, EGR loop etc... Figure 3 shows these ModEngine components. Moreover, the library uses an advanced modelling approach to take accurately into account the relevant physical phenomena taking place in the engine [5]. The computed gas consists of 3 species: fresh air vaporized fuel and burnt gas. Thanks to the object oriented language Modelica, it can be automatically extended to n-gases for future development of the library. Generally, 3 gases is a thermodynamic assumption that has been identified as sufficient for engine simulation results. More than 3 gases, generally 12 gases have to be used for pollutant emissions modelling. Main relevant orifice and pipe elements are available such as air throttle, valve, straight and kneed ducts. Friction and inertial effects can be also taken into account. Heat exchanges are modelled for each element and specific heat exchanger models are also available with air and water cooling systems. An ideal camshaft system is proposed. Inlet and exhaust valves are piloted by valve lift trajectories and cross section characteristics. The elements to build most of the turbocharger technologies are proposed. The modelling approach is based on turbocharger manufacturer's maps.

Concerning the combustion process, a first level of modelling is available with an empirical model based on the Wiebe's law [2]. Generally, this model is used through a mapping of the combustion phenomena based on experimental cylinder pressure; the coefficients of Wiebe's law are calculated defining a map covering the engine operating conditions. This combustion model is a mathematical based model that gives an evolution law for the heat release. 4 coefficients are needed to fit the model on experiment pressure signal for each phase of the combustion. Generally, due to their simplicity, these models are used for real-time applications. The parameters of this function are optimized and mapped according to experimental results. Bohbot *et al.* [8] have used a simple Wiebe law, coupled with an automatic tool to create the parameter maps using both experimental and 3D CFD results. This model can be use either gasoline or Diesel engine simulation. For Diesel simulation, a double Wiebe equation is generally used.

The second level of modelling is given by efficient phenomenological models. For the spark ignition engines, the CFM-1D model is used [3]. This com-

bustion model is based on the CFM combustion model [6], developed at IFPEN in the 3D code IFP-C3D [7]. The coherent flame model (CFM) is a combustion model adapted to the flamelet regime for premixed mixtures. This approach is representative of the premixed flame combustion, which represents the main oxidation mechanism in spark ignition (SI) engines. To calibrate this model, 6 different physical coefficients must be defined to calculate the initialization of the turbulence, the dissipation of the turbulence, the turbulence mixing scale, the flame wrinkling, the flame initial volume and the tumble value. The first 5 are constant coefficients, while the last one defining the tumble coefficient value can be defined as a function of the volumetric efficiency of the engine.

For Diesel engine, an advanced Barba [1] model developed at IFP is implemented in ModEngine [9]. The Barba's model can reproduce the conventional Diesel combustion process, using only 2 zones (a first zone for the description of the pre-mixed combustion and a second one for the diffusion mode). With a reduced number of parameters, it can be used for a wide range of operating points. In this model, the combustion process is divided in 2 steps. In a first step, the fuel is burnt using a premixed model with the hypothesis of flame propagation in the pre-mixed zone. In a second step, when the pre-mixed zone is burnt, the remaining fuel is oxidized using a mixing controlled combustion model. The different hypothesis and equations of the Barba's combustion model are presented in [10].

Cylinder wall thermal exchanges can be taken into account following Woschni models [4]. Injection models allow governing the injected fuel mass rate using maps or algebraic functions. The fuel can be injected in gaseous phase or in liquid phase. The vaporization process is governed by a characteristic timescale for Direct or Port Fuel Injection.

As shown in Figure 3, the ModEngine library contains 22 different packages and at least 250 different models. Figure 4 shows a direct injection single-cylinder modelling and a Mean Value Engine Modelling with the ModEngine library. All models have been validated with dedicated test cases to ensure the non regression of each component (Figure 5) and with functional tests to validate the whole library. The validation of elementary sub models has been done using as reference results the IFP-Engine library developed by IFPEN in the LMS Imagine lab Platform AMESim, functional validation of com-

plete engines, comparison with experimental data, steady state and transient data from test campaigns made at IFPEN.

Figure 2 shows a numerical comparison obtained with ModEngine and IFP-Engine using the Diesel combustion model (Barba) on a one-cylinder Diesel Direct injection engine that validates the good implementation of the Barba model in the Modelica language.

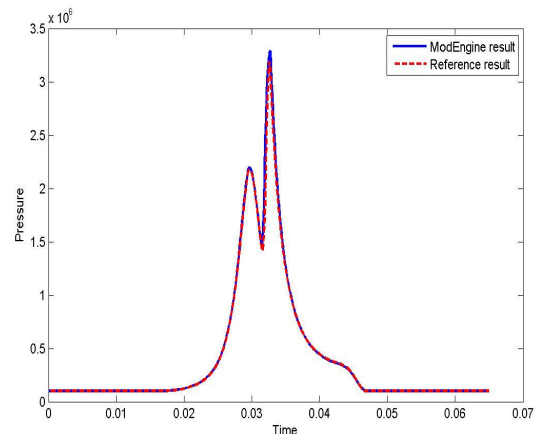


Figure 2: In-cylinder pressure comparisons obtained with Barba Model between ModEngine / Reference (IFP-Engine).

To connect different components, 9 connectors have been implemented in ModEngine. These connectors allow the connection of mechanical part, liquid flow, gaseous flow and thermal flow.

For instance, the connector which links the air path to the cylinder chamber is the PFlowPort connector. Enthalpy and mass flow rate with the mass fraction are defined as input and the output are containing the thermodynamic state and the partial densities.

```
connector PflowPort "Input Pflow Port"
  parameter Integer nGas = 3 "Gaz number";
  input SI.EnthalpyFlowRate dh "input enthalpy flow rate";
  input Real dm "input mass flow rate [kg/s]";
  input Real x[nGas] "input mass fraction vector [null]";
  output SI.Temperature temperature "output temperature";
  output SI.Pressure pressure "output pressure";
  output Real rhoOut[nGas] "output density vector [kg/m**3]";
end PflowPort;
```

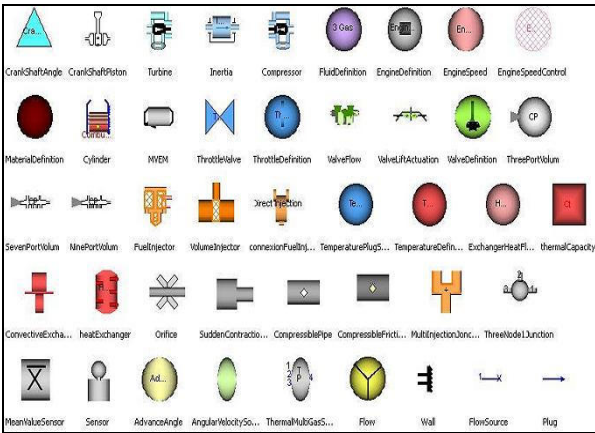


Figure3: ModEngine library

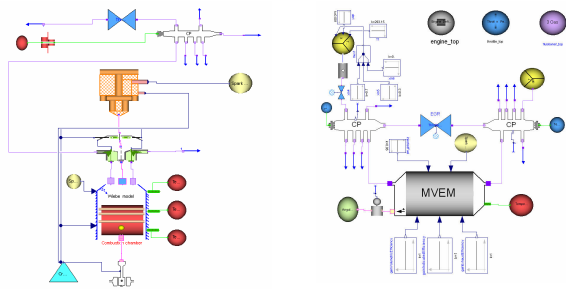


Figure 4: Single cylinder engine and MVEM engine models.

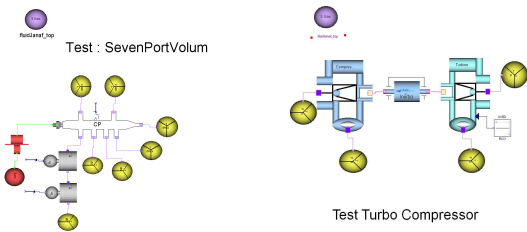


Figure5:

Non regression test case for each component

3 Control development

For the development of engine control strategies, models representing the complete engine are needed. The ModEngine library described above can be used to design such models in Dymola. This first subsection below shows validation results for a complete engine model.

Furthermore, the multiplatform capabilities of Modelica language can be very interesting from a cost point of view, because it allows using a different environment for the execution of the model at this

stage, than that used for the model design. In the following subsection the library results obtained with different platforms are compared.

3.1 Complete engine model validation in Dymola

The engine considered here is a four cylinder gasoline engine with fixed geometry turbocharger. Its model is shown in figure 6. The following approach has also been undertaken for Diesel engines, though it is not exposed here for lack of space.

Two types of tests have been performed: steady state and transient. In steady state the model results are compared with experimental measurements for operating points covering the whole engine range. For transient tests the comparison is made on a driving cycle measured on the test bench.

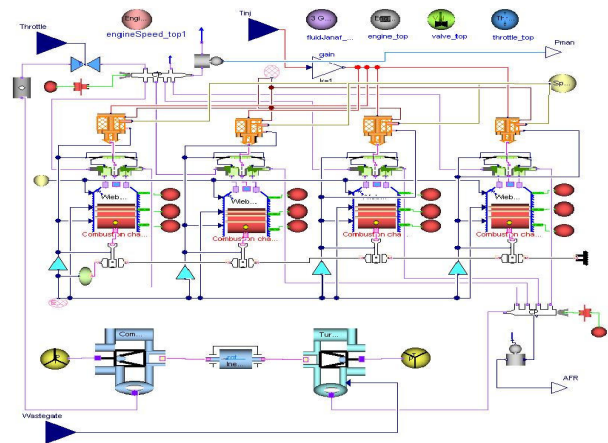


Figure 6 : complete engine model in ModEngine

Steady State tests

The model results are compared for various operating points with experimental measurements. These operating points are defined by :

- engine speed
- intake manifold pressure

The following variables are controlled to setpoints :

- intake manifold pressure is controlled by either the throttle or the wastegate to the setpoint defined by the operating conditions
- the air fuel ratio is controlled at stoichiometry by the injected fuel mass

The thermodynamic conditions along the air system and cylinder are model outputs. The following figure shows a good match between the torque obtained experimentally and with the model. A comparison of

the thermodynamic conditions along the air system would show similar results.

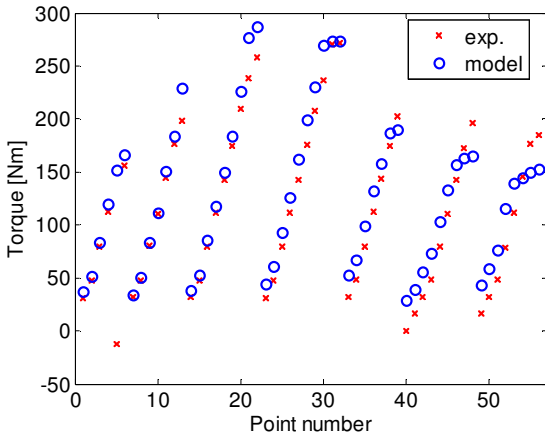


Figure 7: The torque match

Transient tests

The model is now compared with transient results. It is plugged to engine control software, running in co simulation in Simulink. The inputs of the model are:

- engine speed
- torque setpoint

The engine control software determines in closed loop the commands of the engine (actuators positions: throttle and waste gate, injection timing, spark advance) based on the sensor values received from the model. The results are shown in the figure 8, enhancing again the good behavior of the simulator with respect to experiments.

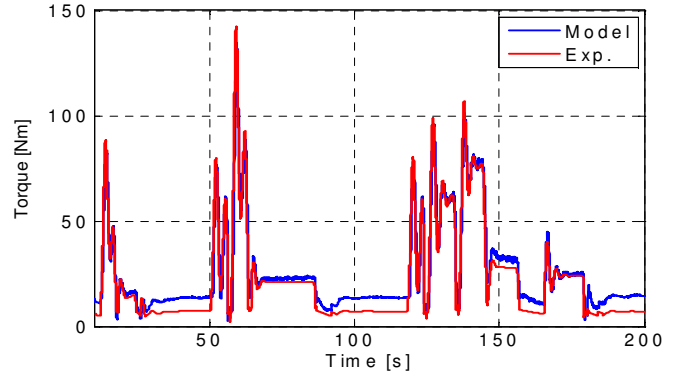
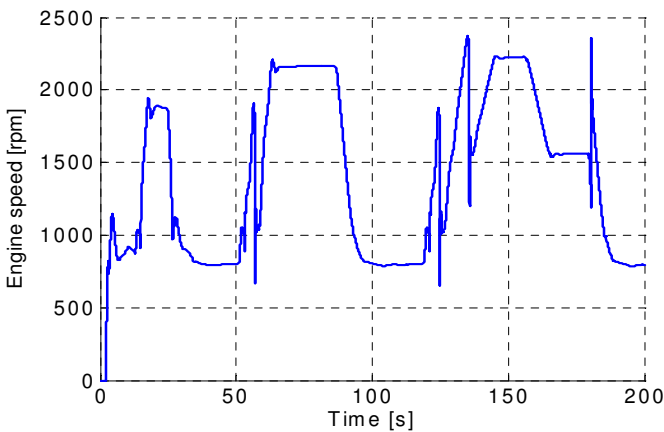


Figure 8

3.2 Comparisons: Dymola/Openmodelica

All the elementary models developed in Dymola were translated in Openmodelica. The translation was not immediately done and needed some reshuffle. Comparisons used a fixed and variables step integrators (Runge-Kutta / Euler; Dassl, /Dassl2).

The main variables which were selected to verify the accuracy of the results using Dymola and OpenModelica are the temperature, the pressure in the cylinder (figures 9, 10, 13), and the Dissipative kinetic energy (figures 11, 12, 14).

We show following some comparisons between Dymola and OpenModelica for 3 combustions models.

Wiebe model

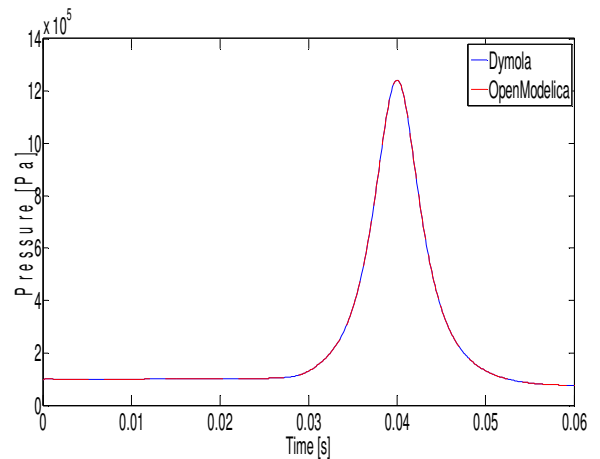


Figure9: Cylinder Pressure

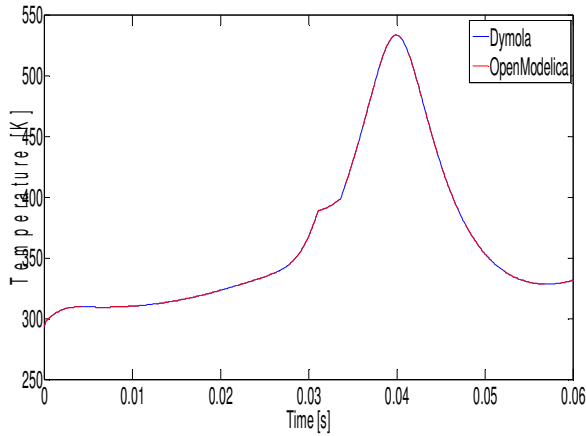


Figure10: Cylinder temperature

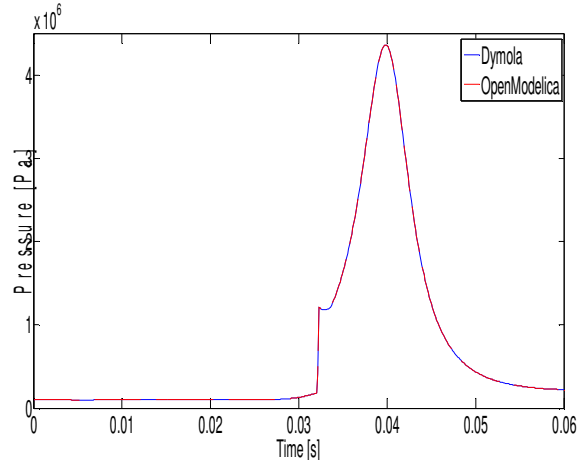


Figure13: Pressure model

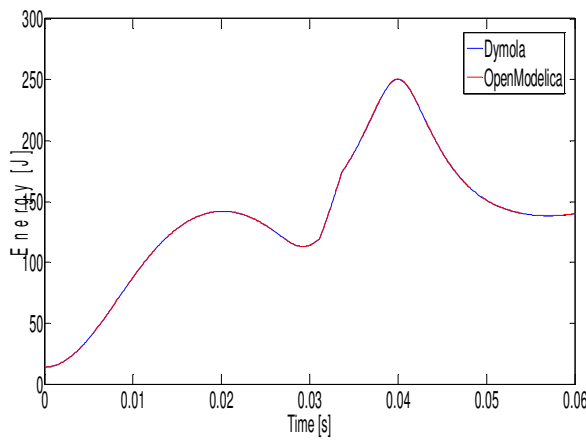


Figure11 : Energy model

Barba model

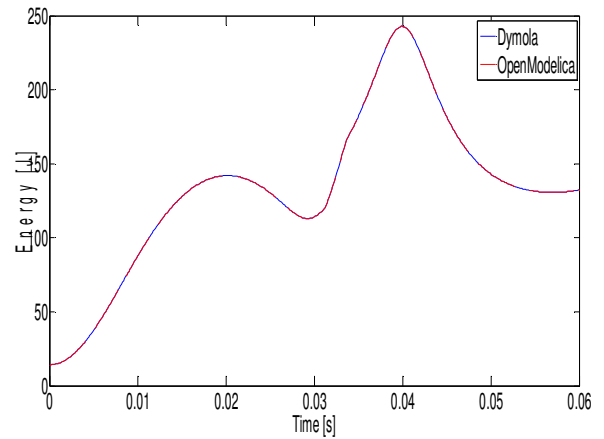


Figure14: Energy model

CFM model

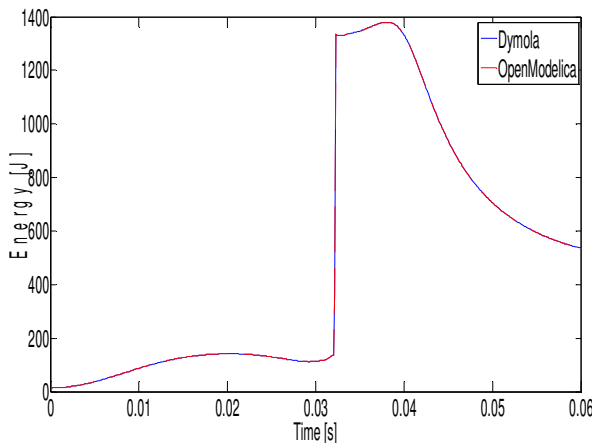


Figure12: Energy model

3.3 Conclusions

ModEngine simulation results are performed respecting the same integration conditions using the two platforms Dymola and OpenModelica.

Models contain 683 variables and equations; with 235 zeros crossing and without numerical jacobians.

For the same final time simulation and integrators having the same tolerances it seems that Dassel and RungeKutta in Dymola take acceptable equivalent time, while integration with OpenModelica with both fixed or variable step takes more longer time.

The two platforms give similar results; the errors are less than 1%.

4 System Integration and control validation

The multidisciplinary aspect of complex systems leads to use different tools for the design step. That's why the simulation step requires co-simulation techniques in order to exchange data between simulators. The concept of functional mockup provides advancement to model exchange during the product design and validation cycles. Using models through their interfaces allows hiding their implementation details and making their usage easier.

4.1 xMOD

xMOD [12] is a platform which combines an integration environment for various heterogeneous models, together with a virtual test laboratory. xMOD offers a range of different functionalities, such as the integration of heterogeneous models (Simulink, AMESim...), confidentiality management for models when they are imported, virtual instrumentation, test automation, etc. The purpose of xMOD is to make it possible for models to be used by people other than those who created them, and for them to be shared. xMOD provides simulation functions in various simulation schemes: real-time, extended time or as soon as possible. Its execution kernel can be used to process various integrated models in multiprocessor and multicore. xMOD is built around the following key ideas:

- Using a unified representation of all heterogeneous models that is simple and complete enough for them to be integrated and co-simulated, and for the expertise that they contain to be protected.
- Abstracting the modelling language through a virtual instrumentation, such that the models can be easily understood by people other than those who created them, or by people who do not have knowledge of the languages in which they were written.
- Focusing on using the models (they are always built in the usual modelling environments), and providing ergonomically-designed features for interacting with the simulations, running the tests and using the results.

4.2 FMI

The ITEA2 project MODELISAR is providing solutions enabling the integrated design, test and management of automotive systems. One result of this

project is a new open Functional Mockup Interface (FMI) to support co-simulation between simulation tools, in particular Modelica, for system modelling and AUTOSAR for embedded control software generation [24]. The FMI specifies C and XML interfaces for dynamic systems to be used as an interchange format between different tools. This interface is to be implemented by an executable called FMU (Functional Model Unit). The FMI functions are called by a simulator to create one or more instances of the FMU, called models, and to run these models, typically together with other models. An FMU may either be self-integrating (co-simulation) or require the simulator to perform numerical integration. The FMI goal is to describe models of dynamic systems which are, in general, described by differential, algebraic and discrete equations with time, state and step events. The interface is designed so that large models can be described and consists of the following two parts: A model interface: All needed equations are evaluated by calling standardized C functions. A model description schema: All variables in the model are defined in a standardized way in a XML file. The C-code could then be executed in an embedded system without the overhead of the variable definition.

4.3 Integrating FMI in xMOD

In order to extend the capabilities of our co-simulation tool we chose to integrate the FMI functionalities to support more heterogeneous models coming from Modelica based tools. This extends the capabilities of xMOD allowing it to integrate models (in the form of FMUs) coming from various Modelica compatible authoring tools like AMESim, Dy-mola, SimulationX, Simpack...

To integrate FMI-for-cosimulation functionalities in xMOD, we opted for the wrapper approach. This solution is applicable for tools offering library interfaces with the ability to call functions or methods. In xMOD, each instantiated model has its library interface providing common generic functions to evaluate the model dynamic equations. To integrate FMI functionalities, we chose to develop a wrapper library whose main purpose is to load FMU models and call their FMI functions. All the FMU model functions are wrapped in the common generic functions. The class diagram below shows the part of the design of the FMU wrapper. The common generic functions are listed in the xMODFMU wrapper class. Based on this approach, we succeed to make xMOD FMI compatible.

4.4 Control validation

Before their implementation in the actual electronics unit, it is necessary to validate the control strategies in a "real-time representative" simulation environment. Thanks to xMOD and to the implementation of the FMI concept, it is possible to build global co-simulations involving the physical models (from ModEngine), imported in xMOD as FMUs, as well as the control strategies, which are usually developed in Simulink, and which are imported in xMOD using the xMOD Target for Real-Time Workshop [14]. This global co-simulation may be used for control laws parameters tuning and pre-calibration as well as closed loop system performance assessment. (Figure 15).

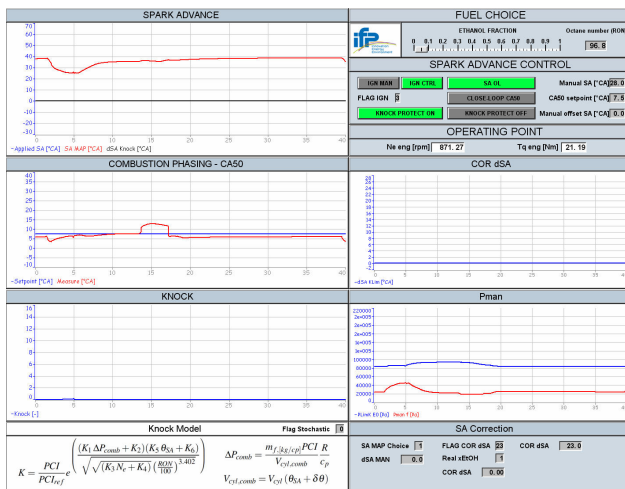


Figure 15: Screen shot from xMOD execution window illustrating the validation of a co-simulation platform integration an FMU of a flex-fuel combustion engine, which was generated from Dymola, running together with Simulink models (controls) and AMESim models (vehicle Dynamics)

5 Perspectives: real-time simulation

Real-time simulation of a single model is a non-sense. Real-time simulation is needed when models are supposed to interact with physical part of the whole system, for example when coupling simulated and real components in a Hardware-in-the-Loop (HiL) process. Indeed, this process is representative and successful only if components are unable to distinguish if others components are real or simulated. It implies that a simulated component must have the same timing behaviour than its corresponding real component. Real-time constraints are consequently inherited from the needed data exchange between components. For example, let consider we want to

connect our engine models to a real hardware controller which acquire sensor data and send its actuators command every 500µs. To make possible this HiL process we must ensure that our model simulation accuracy is sufficient to always be able to simulate 500µs of engine behaviour in less than 500µs of real-time. Notice that verifying that you can simulate 30 seconds of the engine behaviour in less than 30 seconds of real-time is not sufficient to guarantee the previous requirements. Indeed, in the HiL process, even if 500µs of engine behaviour are simulated in less than 500µs, the engine simulation cannot go on before the end of the 500µs period in order to receive controller data.

Consequently, improving performance for Madelia models is a necessary condition for simulations to reach real-time. Engineers often think about improving their models efficiency at the end of the design phase. This leads often to non efficient simulations. In [13] we showed that the needs for efficiency should be considered as soon as the modelling step starts. A set of general methods based on a closer view on the Modelica's modelling and simulation processes are presented to give hints to the designers in order to reach real-time requirements.

The Functional Mock-up Interface could also help to gain in efficiency for Modelica models. Indeed, the FMI and especially, the FMI for Model Exchange, gives freedom to the user to handle model's execution in different ways. For example, the FMI does not enforce any predefined event handling mechanism like the one provided by Dymola or other Modelica tools.

6 Modelica contribution and future works

We end this article on the observations and reflections on the use of language and focused on a return of the technical problems [15] that in fact open to other possibilities to extend the language.

Today we are participating intensively in various activities on Modelica. Among the areas for future work , we will give the possible directions we intend to take .

6.1 Dymola and OpenModelica implementation feedback

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

for research, teaching and industrial usage. However, we think that for this latter case, Dymola is ahead of OpenModelica tool. Until today, we notice that OpenModelica doesn't support yet all Modelica specifications. Thereby, the original models have been to be depreciated to run correctly. We noticed also that the OpenModelica fixed step solver is much accurate then Dymola fixed step solver. For example, the figure 15 compares the RungeKutta 4 solver of the two platforms against the Dassl solver which is taken as the reference:

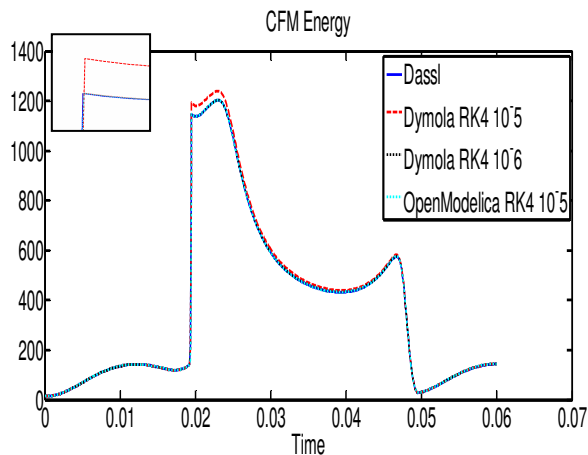


Figure 15

Finally, we can notice also that Dymola provides unique support for real-time and hardware-in-the-loop simulation (HILS) instead of the actual version OpenModelica.

6.2 Future directions

LMS [31] and INRIA [27] have developed a compiler based on Modelica, named Modelicac[18], during the SIMPA2-C6E2 project [16, 17], which is still in progress. Our aims in the near future is to both continue to test Modelica langage on different platform AMESim/ScicosLab, OpenModelica and Dymola, in order to have heterogenous, interoperable and multiplatform librairy. We will continue our involvement in European projects, more particularly on the control, FMI , xMOD and real-time applications involving Modelica language.

Key elements of the long-term orientation of our work is the success of European projects. Indeed, if it turns out that Modelica provides benifit and is the "standard" recognized by the industry, several fields of engineering systems will adopt Modelica language.

References

- [1] Barba C., Burkhardt C., Boulouchos K., Bargende M. (2000) A phenomenological combustion model for heat release rate prediction in high speed DI Diesel engines with common rail injection, *SAE Technical Paper* 2000-01-2933.
- [2] Wiebe, I.I., "Semi-empirical expression for combustion rate in engines", *Proceedings of Conference on piston engines, USSR Academy of sciences, Moscow, pp. 186-191, 1956.*
- [3] F-A. Lafossas, O. Colin, F. Le Berr, P. Menegazzi, "Application of a new 1D combustion model to gasoline transient engine operation", *SAE 2005 Fuels and Lubricants Meeting Exhibition and Congress, May 11-13 2005, Rio de Janeiro, Brazil - SAE 2005-01-2107.*
- [4] Woschni G., "Universally Applicable Equation for the Instantaneous Heat Transfer Coefficient in the Internal Combustion Engine", *SAE paper 670931, SAE Trans., vol. 76, 1967.*
- [5] Menegazzi, P., Aubret, P., Vernhes, P.-L., "Conventional and Hybrid Vehicle Emission, Fuel Economy and Performance Analysis System Simulation", *FISITA 2004, 23-27 May, Barcelona, Spain*
- [6] Colin, O., Benkenida, A., Angelberger, C., "A 3D Modelling of Mixing, Ignition and Combustion Phenomena in Highly Stratified Gasoline Engines", *Oil & Gas Science and Technology, vol. 58, pp. 47-62, 2003*
- [7] "IFP-C3D: an Unstructured Parallel Solver for Reactive Compressible Gas Flow" J. Bohbot, N. Gillet, A. Benkenida, *Oil Gas Sci. Tech., 64 (2009), 309-336*
- [8] Bohbot J., Lafossas F.-A., Miche M., Chraibi M., Menegazzi P. (2004) A new coupling approach using a1D system simulation software and a 3D combustion code applied to transient engine operation, *SAE Technical Paper* 2004-01-3002.
- [9] F.-A. Lafossas, M. Marbaix and P. Menegazzi "Development of a Coupling Approach between 0D D.I. Diesel Combustion and Pollutant Models: Application to a Transient Engine Evolution" *Oil & Gas Science and Technology - Rev. IFP, Vol. 63 (2008), No. 4, pp. 479-494*

- [10] Barba C., Burkhardt C., Boulouchos K., Bargende M. (2000) A phenomenological combustion model for heat release rate prediction in high speed DI Diesel engines with common rail injection, *SAE Technical Paper 2000-01-2933*.
- [11] Modelisar Functional mock_up interface for model exchange version 1.0. Technical Report 07006, *MODELISAR consortium, January 2010*.
- [12] M. Ben Gaïd, G. Corde, A. Chasse, B. Léty, R. De La Rubia, M. Ould Abdellahi. Heterogeneous Model Integration and Virtual Experimentation using xMOD: Application to Hybrid Powertrain Design and Validation In Proc. *7th EUROSIM Congress on Modeling and Simulation, Prague, Czech Republic, September 2010*.
- [13] 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.
- [14] Coppin Thomas, Grondin Olivier, Le Sollic Guenaël, Maamri Nezha, Rambault Laurent. "Control-oriented mean – value model of a fuel-flexible turbocharged spark-ignition engine". SAE World congress, Detroit USA, 13-15 april 2010.
- [15] Benjelloun Zakia; Moulin Philippe, Najafi Masoud, Shen Xduong. "Simulation of the mean-value internal combustion engine in modelica" MathMod International Conference on Mathematical Modelling, Vienna, Austria, 11-13 February 2009.
- [16] Benjelloun Zakia, Najafi Masoud. "Using modelica for modelling and simulation of spark ignited engine and drilling station in IFP". International modelica conference, Bielefeld, Germany, 3-4 march 2008.
- [17] Benjelloun Zakia, Najafi Masoud. "Modelling complex system with modelica in scicos: application to mean value spark engine". ESM European Simulation and Modelling conference, St.Julian's, Malta, 22-24 October 2007.
- [18] Masoud Najafi, Ramine Nikoukhah, Serge Steer, Sebastie Furic. "New features and new challenges in modelling and simulation in Scicos" Proceedings of the 2005 IEEE Conference on Control Applications. Toronto, Canada, August 28-31, 2005.
- [19] <http://www.modelica.org/>
- [20] <http://www.3ds.com/products/catia/portfolio/dymola>
- [21] <http://www.openmodelica.org/>
- [22] <http://www.ifpennergiesnouvelles.fr/>
- [23] <http://www.eurosyslib.com/>
- [24] <http://modelisar.org/>
- [25] www.openprod.org
- [26] <http://www.amesim.com/>
- [27] <http://www-rocq.inria.fr/scicos/>
- [28] <http://www.pme.gouv.fr>
- [29] <http://www.itea2.org/>
- [30] <http://www.ida.liu.se/labs/pelab/modelica/OpenSourceModelicaConsortium.html>
- [31] <http://www.LMSINTL.com>

Thanks

This work was realized thanks to the labeling of the projects Eurosyslib, Modelisar, and OpenProd by the European Organisme ITEA 2 (Information Technology for European Advancement)[29] and, thanks to financial support from DGCIS (Direction Générale de la Compétitivité, de l'Industrie et des Services)[28].

IFPEN is an OSMC [30] member and thus we benefits from fruitful discussions during the adaption of the library ModEngine in OpenModelica

UN-VirtualLab : A web simulation environment of OpenModelica models for educational purposes

Oscar Duarte

Universidad Nacional de Colombia, Department of Electrical and Electronics Engineering
ogduartev@unal.edu.co
Carrera 30, Calle 45, Ed. 453, Of. 202. Bogotá, Colombia

Abstract

In this paper a free web simulation environment is presented: UN-VirtualLab . It is a virtual laboratory in which users can perform experiments on precompiled software models. UN-VirtualLab uses OpenModelica in order to compile software models written in Modelica language. The main features and internal architecture of the system is presented. Some potential applications are discussed.

Keywords: simulation environments, web applications, OpenModelica, education

1 Introduction

UN-VirtualLab is a web simulation environment distributed under GPL license. It is a virtual laboratory in which users can perform experiments. Using a web browser, any regular user can pick up an experiment, modify its parameters and simulate its response. The plant in which the experiment is done does not physically exist, it is a pre-compiled software model. UN-VirtualLab uses OpenModelica in order to compile software models written in Modelica language.

The term ‘virtual laboratory’ refers, in a broad sense, to an electronic and software workspace for experimentation. Many different approaches satisfy this definition. In order to clarify what kind of them UN-VirtualLab is, we may use some classification criteria:

- *Web aviability:* Some virtual labs are installed in a PC as a local software package where as others are installed in a web server. Usually, in the second approach the user connects with the server through a conventional internet navigator. In this paper we use the term ‘virtual’ for web based tools, such as UN-VirtualLab .
- *Physical vs. software models:* In some virtual labs the plants in which the experiments are con-

ducted physically exist ([1], [2]) where as in others they are software models([3],[4]). In the first approach there are sensors and actuators connected with the virtual lab, and the user can manipulate them. In UN-VirtualLab the plants are software models.

- *Interactivity:* In [5] a distinction is made between two types of software model based Virtual labs: those which allow the user to perform actions during the simulation and those which not. The first approach is refered as *runtime interactivity* and the second one as *batch interactivity*. UN-VirtualLab has batch interactivity.
- *Subject orientation:* Some virtual labs are designed to satisfy an specific need ([4], [6]) where as others are of general purpose ([7]). In the first approach, the solution is not easy to use in a different context; for example, a virtual lab for biological plants can not be converted to a virtual lab for mechanical plants. UN-VirtualLab is of general purpose.
- *Course orientation:* Some virtual labs are course oriented ([3]) where as others are not. In the first approach, there are tools as student and grade management, individual and group progress reports, etc. UN-VirtualLab is not course oriented.

Some efforts have been made to combine Modelica and virtual labs: In [8] a web version of the well known DrModelica software is presented. In [9] and [5] virtual labs are developed by using a combination of Dymola and Sysquake. In [10] a Modelica-based algorithm is developed to implement the interactive mechanism. In [11] a web service is developed to compile and simulate remotely Modelica based models.

In this paper we show a simple and different approach, based on OpenModelica. OpenModelica is

an open source modeling and simulation environment ([12], [13]). Using OpenModelica it is possible to compile Modelica models. Once the compilation is done, an *executable file* is available. When running, the executable reads an *input file* and writes a *results file*, as shown in figure 1.

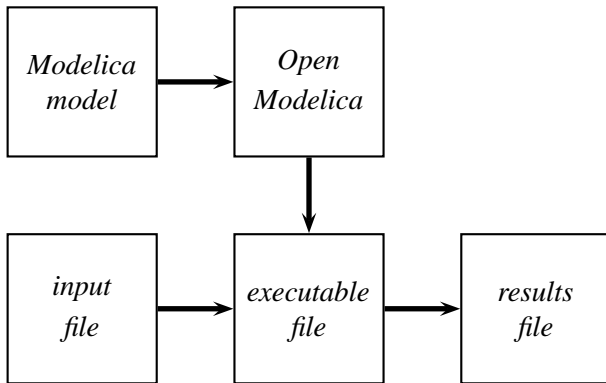


Figure 1: Files in an OpenModelica simulation.

UN-VirtualLab interacts in a web environment with these files in the following way:

- Brings a graphical interface to modify selected parameters of the *input* files.
- Runs the *executable* file.
- Displays the data of the *results* file using plots, tables as well as 2D and 3D animations.

In this paper we summarize the main features of UN-VirtualLab (section 2) and describe its internal architecture (section 3). We also discuss possible applications in section 4. Conclusions and future work are presented in section 5.

2 Features

In UN-VirtualLab experiments are organized by nested subjects. It is possible to define a tree of subjects and include any number of experiments in every subject.

The same Modelica model can be used in several experiments. As an example, using the same model of an electrical vehicle it is possible to design an experiment to analyze de controller performance, another to perform sensitivity analysis of the vehicle mass, and another one to study power consumption.

Once a user selects an specific experiment, he/she can:

- Modify the parameters of the experiment. The parameters are those defined in the *input file*. For convinience, they are classified in three types:

Simulation parameters: such as *start time*, *stop time*, *step value*, *tolerance* and *method of integration*.

Initial conditions: start value of simulated variables.

Model parameters: any other parameter in the *input file*. These parameters can be arranged in groups.

- Simulate the model and visualize the results. There are up to four options of visualization:
 - Plots.
 - 2D animations.
 - 3D animations.
 - Data tables.
- Read or download the experiment documentation, including:
 - Model description.
 - Modelica source code.
 - Author information.

Other features are:

- Multilanguage support, selected by the administrator.
- Easy appearance customisation, using css themes.

3 Architecture

UN-VirtualLab is written in php language. Modelica models, *input*, *executable* and *results* files of every experiment are stored in individual directories. Every single experiment is defined by an *xml file*. System administrator can modify these xml files using a graphical interface (see figure 2).

When a user picks up an experiment, the system reads the corresponding *xml* file and creates a graphical interface that shows the outputs of the simulation with the default values of the parameters. It also shows a form so the user can change these values and launch a new simulation (see figure 3).

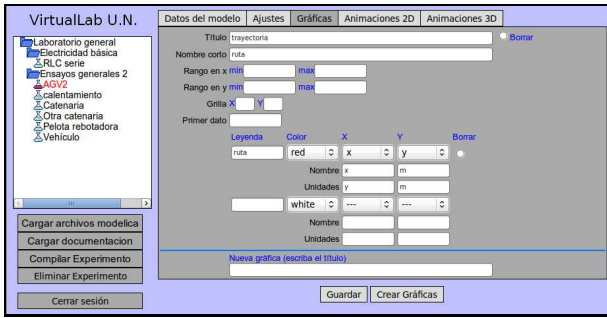


Figure 2: Administrator interface

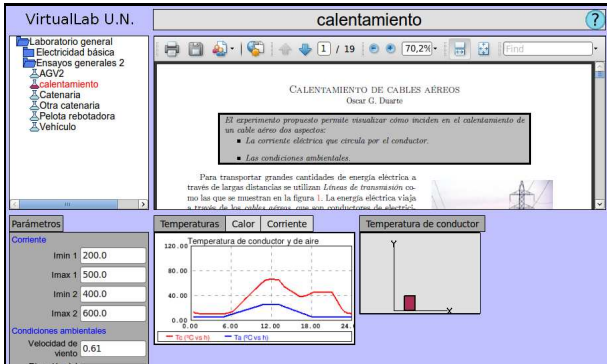


Figure 3: Experiment interface

Figure 4 shows how UN-VirtualLab processes a simulation order. When the simulation is launched, the actual values defined by the user are read and a *temporary input file* is created, the system runs the *executable file* and creates a *temporary results file* which is used to generate on line plots, animations and tables. Then, temporary files are removed.

Notice that the same structure shown in figure 4 can be used with any executable file that uses input and results files, not just by OpenModelica compiled files. In that sense, UN-VirtualLab has a very general structure and can be used with a broad spectrum of simulation packages. However, UN-VirtualLab actually recognizes just the input and results file formats used by OpenModelica.

3.1 Plots and tables

Curves to be plotted are defined by selecting a pair of simulated variables. Usually, the first one is *time* but not necessarily, so it is possible to plot two-dimensional phase portraits. As an example, consider the bouncing ball plant, in which *h* represents the height of the ball, *v* its velocity and *t* the time. We can plot (*h* vs. *t*), (*v* vs. *t*) and (*h* vs. *v*).

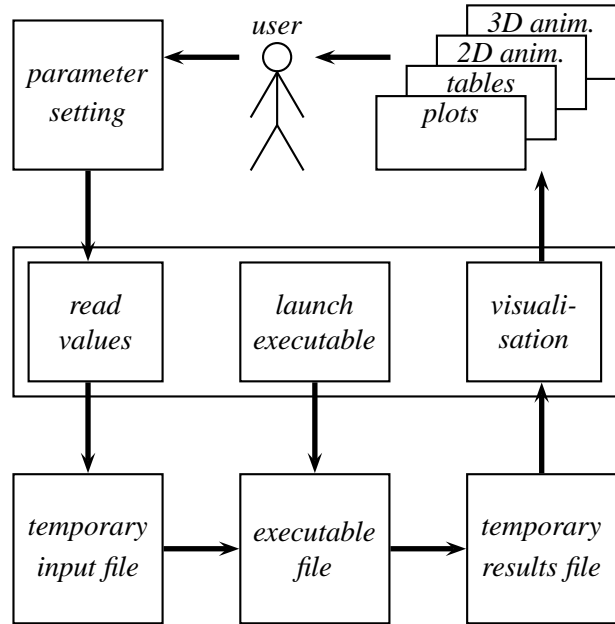


Figure 4: Files in a UN-VirtualLab simulation.

Plots are made on line taking the values written in the *results* file. A php class has been written to produce the images that are displayed using the png format. Using the same plot more than one curve can be drawn, as figure 5 shows.

The data plotted is also available in data tables. User can download the data of his own simulation as plain text. Columns are separated by <TAB> character, so it is possible to import directly the downloaded data into an spreadsheet as OpenOffice Calc.

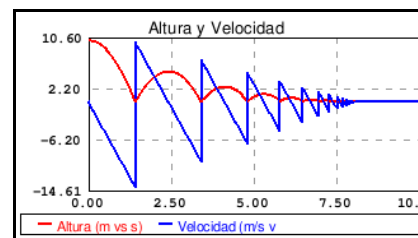


Figure 5: Plot example. Height and velocity of a bouncing ball.

3.2 2D animations

2D animations are based on some primitives whose properties are changed by the values stored in the *results* file. The primitives available are: axis, rectangles, ellipses, rings and polylines. The properties that can be driven by the simulation results are: size (*x* and

y), position (x and y) and rotation (around z axis, orthogonal to the animation plane).

Following with the bouncing ball example, a 2D animation can be made with two primitives: a rectangle for the floor, and a circle for the ball (see figure 6). The y coordinate of the circle position can be driven by the h variable in the *results file*.

2D animations of UN-VirtualLab are animated png/gif files. In order to build every frame, several php classes have been written. The combination of the individual frames in a single png or gif animated file is done with the *apng-creator* and *dgifanimator* libraries respectively [14]. The number of frames and the time between frames of every animation can be adjusted. It is also possible to configure the ($x - y$) position of the camera and an scale factor.

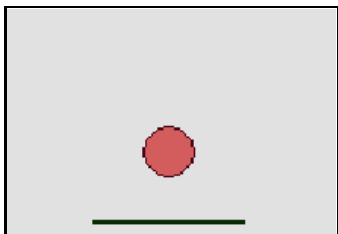


Figure 6: 2D animation snapshot. Height of a bouncing ball.

3.3 3D animations

3D animations are made in a similar way to 2D animations. Primitives available are: axis, cubes, spheres, pipes and cylinders. The properties that can be driven by the simulation results are: size (x , y and z), position (x , y and z) and rotation (around x , y and z axis).

Figure 7 shows an snapshot of a 3D animation of the bouncing ball. It has been made with two primitives: a cube for the floor, and an sphere for the ball (see figure 6). The z coordinate of the sphere position is driven by the h variable in the *results file*.

As in 2D animations, the number of frames and the time between frames of every animation can be specified by the system administrator. It is also possible to configure the (x, y, z) position of the camera, (x, y, z) focus point and an scale factor.

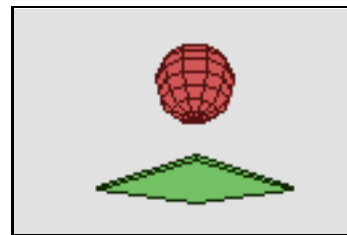


Figure 7: 3D animation snapshot. Height of a bouncing ball.

3.4 Documentation

UN-VirtualLab displays information about the experiment. The experiment author must prepare this information as pdf files. UN-VirtualLab uses *pdftohtml* to produce the html files from the pdf.

Authors can use \LaTeX and a suggested \LaTeX style to produce the pdf files. The suggested style has a customisation of the *listings* package ([15]) for publishing Modelica source code. It recognizes a subset of the Modelica language specification, often enough to produce fancy documentation files (see figure 8).

File 1: Asinh.mo

```

within Catenary ;
function asinh
  input Real x;
  output Real y;
  external "C" y=asinh(x);
end asinh;

```

Figure 8: Example of \LaTeX output of Modelica source code using the suggested style.

3.5 Layout and appearance

The user interface has five blocks, as shown in figures 3 and 9. They are:

1. Experiment selection block: a tree menu to pick up the subject and experiment.
2. Parameter settings block: a dialog form to change de default values of the choosen experiment parameters.
3. Documentation block: a frame display the pdf/html experiment documentation, and the links to downloadable files (Modelica source code and documentation files).

4. Plots and tables block: two frames to display the simulation results as plots and data tables, respectively.
5. Animations block: some frames to display the simulation results as 2D and 3D animations.

The appearance of the interface can be changed using different css themes. Not only colors and fonts can be changed, but also the size and position of the blocks.

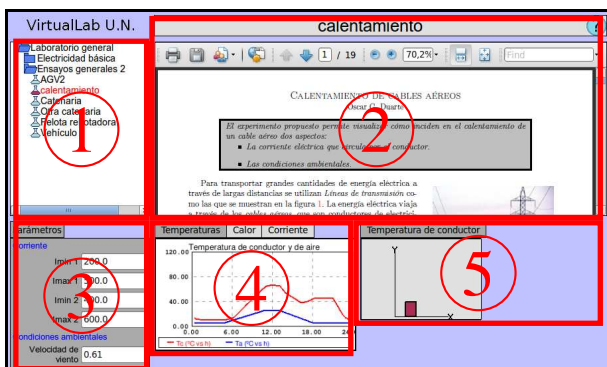


Figure 9: Layout

1. Publishing research results: suppose you have finished a research project and as a result you have a novel dynamic model of something. You have published good papers in recognized journals, but you also want to explain the results to a wider public. You may use UN-VirtualLab to let the visitors of your web site to explore your model.
2. Novice students laboratories: suppose you are in charge of a first year course in an engineering program. You want your students to know something about more advanced topics, perhaps just to help them to understand some basic concepts. You want them to experiment with some plant, but they do not have yet enough skills and knowledge neither to do it in real life nor to write a simulation software. You may use UN-VirtualLab to bring them a convenient simulation environment.
3. Complement traditional teaching: the benefits of simulation environments in traditional teaching have been widely reported (see, for example [16]). Using UN-VirtualLab it is possible to design some experiments that help students to explore more aspects of a concept than those explored in the classroom.

3.6 Multilingual support

UN-VirtualLab has multilingual support. Actually there are two languages available: English and Spanish. Two different aspects has been addressed to implement multilingual support:

- Common interface: all the strings that are common to all experiments. They are defined in a single file.
- Experiment data: every single experiment has specific strings as experiment name, parameters names, plots titles, etc. These strings are defined in the *xml file*. It is possible to define different *xml files* for the same experiment, each one for a different language.

4 Possible applications

UN-VirtualLab is not intended to replace simulation tools as Dymola, SimulationX or OpenModelica. The main purpose of UN-VirtualLab is to publish simulation experiments on the web. Some potential applications are the following:

5 Conclusions and future work

UN-VirtualLab provides a light web simulation environment for pre-compiled software models. Using web simulation environments, many people can access the same simulation engine and the licenses costs is reduced. Using UN-VirtualLab and OpenModelica, it is possible to implement a totally free software solution.

There are some aspects that must be reinforced in the short term:

- As stated in section 3, even that UN-VirtualLab internal structure is very general, actually it recognizes just the input and results files produced by OpenModelica. More formats should be recognized.
- The animations can be more complex, by driving more primitive properties such as colors and line widths. The use OpenGL and other graphical libraries must be explored.
- It is important to research how to implement interactive simulations. The use of the interactivity option of OpenModelica through the web is not trivial, but must be explored.

The first public application of UN-VirtualLab is available at the Virtual Academic Services of the National University of Colombia ¹. According with the actual schedule, in June of 2011 it will include at least a hundred of experiments, most of them from engineering subjects.

References

- [1] R. Pastor, C. Martín, J. Sánchez and S. Dormido Development of an XML-based lab for remote control experiments on a servo motor International Journal of Electrical Engineering Education. Vol 42 No.2 2005 pg 173- 184
- [2] Paul I-Hai Lin and Melissa Lin Design and Implementation of an Internet-Based Virtual Lab System for eLearning Support Proceedings of the Fifth IEEE International Conference on Advanced Learning Technologies (ICALT'05) 05-08 July 2005, Kaohsiung, Taiwan.
- [3] Dongil Shin , En Sup Yoon, Kyung Yong Lee, Euy Soo Lee A web-based, interactive virtual laboratory system for unit operations and process systems engineering education: issues, design and implementation Computers and Chemical Engineering 26 (2002) 319– 330
- [4] Diwakar, Achuthan and Nedungadi Biotechnology virtual labs- Integrating wet-lab techniques and theoretical learning for enhanced learning at universities. IEEE. 2010 International Conference on Data Storage and Data Engineering Bangalore, (International), (February 2010)
- [5] Carla Martín Villalba Object-Oriented Modeling of Virtual Laboratories for Control Education Doctoral Dissertation Universidad Nacional de Educación a Distancia Escuela Técnica Superior de Ingeniería Informática Madrid, 2007
- [6] Hu Shaobin, Yin Daiyin, Cao Guangsheng, Guo Lingling Construction and Application of Autonomous Learning Based Remote Downhole Tool Virtual Lab 2010 2nd International Conference on Education Technology and Computer (ICETC) Shanghai, June 2010
- [7] Alfonso Urquía, Carla Martín-Villalba Laboratorios Virtuales Interactivos <http://www.euclides.dia.uned.es/simulab-pfp/index.htm>
- [8] Eva-lena Lengquist S , Susanna Monemar , Peter Fritzson , Peter Bunus DrModelica – A Web-Based Teaching Environment for Modelica In Proceedings of the 44th Scandinavian Conference on Simulation and Modeling (SIMS'2003)
- [9] Carla Martín-Villalba, Alfonso Urquía, Sebastian Dormido Object-oriented modelling of virtual-labs for education in chemical process control Computers and Chemical Engineering 32 (2008) 3176–3186
- [10] Wenbin Jiang, Zhen Tang, Hai Jin, Chao Liu MISM: Modelica-based Interactive Scheduling Mechanism for Virtual Educational Experiments The Fifth Annual ChinaGrid Conference Guangzhou, China, July 16-18, 2010
- [11] Sven Meyer zu Eissen, Benno Stein Realization of Web-based simulation services Computers in Industry 57 (2006) 261–271
- [12] OpenModelica project web site. <http://www.openmodelica.org>
- [13] Peter Fritzson, Peter Aronsson, Adrian Pop, Håkan Lundvall, Kaj Nyström, Levon Saldamli, David Broman, Anders Sandholm: OpenModelica - A Free Open-Source Environment for System Modeling, Simulation, and Teaching, IEEE International Symposium on Computer-Aided Control Systems Design, October 4-6, 2006, Munich, Germany
- [14] PHP Classes Repository. <http://www.phpclasses.org>
- [15] CTAN, the comprehensive T_EXarchive network. Listings package. <http://www.ctan.org/tex-archive/macros/latex/contrib/listings/>
- [16] Zacharias C. Zacharia, Georgios Olympiou Physical versus virtual manipulative experimentation in physics learning Learning and Instruction (2010) article in press
- [17] Dirección Nacional de Servicios Académicos Virtuales. <http://www.virtual.unal.edu.co>

¹<http://www.lab.virtual.unal.edu.co>

Extending the IPG CarMaker by FMI Compliant Units

Stephan Ziegler and Robert Höpler
Modelon GmbH München
Agnes-Pockels-Bogen 1, 80992 München, Germany
{stephan.ziegler, robert.hoepler}@modelon.com

Abstract

This paper presents a generic interface which enables exploiting the multi-physics modeling capabilities of Modelica within the virtual test drive simulator CarMaker [1] using the Functional Mock-up Interface (FMI) [2].

Applications ranging from detailed studies of vehicle properties to hardware in the loop test-rigs require models of different complexity. CarMaker is provided an interface which allows for vehicle models being extended by almost arbitrary external component models implementing the FMI. The FMI offers two flavors of units which permit the balancing of computational performance and numerical robustness within CarMaker. Additional model information provided by the FMI is used for automatic integration of the external models as well as for configuring the computations. Involved aspects are shown by an example truck model.

Keywords: FMI, FMU, CarMaker, HIL, Solver, Model export, Interface, Stability, Co-Simulation

1 Introduction

The CarMaker platform [3] is a full-fledged virtual driving environment which offers a wide range of applications from offline operation to hardware in the loop (HIL) tests. CarMaker was designed to support the development process from an early conceptual stage to hardware prototype testing.

Therefore the CarMaker suite is composed of two main components, the CarMaker Interface Toolbox (CIT) and the Virtual Vehicle Environment (VVE), see Fig. 1. The CIT contains a collection of tools for simulation control, parameterization, analysis, visualisation, and file management.

The VVE represents the computer modeled composition of the vehicle with all its components such as powertrain, tires, brakes and chassis as well as

road and driver. Vehicle components may be implemented by default generic models, custom code such as MATLAB/Simulink controller models or even real hardware on a test rig. Depending on the desired task

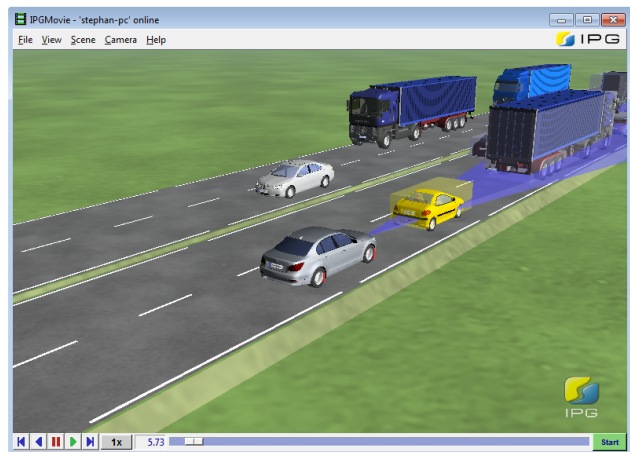


Figure 1: Typical VVE scenery including road and traffic rendered by the CarMaker IPG-Movie animation tool.

the VVE can be operated on a regular office computer or on a real-time system. Real-time operation allows investigation of deterministic behavior, office operation might lack real-time capabilities but is therefore applicable on almost any host computer and allows the simulation to run slower or faster than real-time depending on system performance and model complexity and does not require special hardware.

Simulation techniques are a key enabler in the development of nowadays propulsion systems of electric and hybrid vehicles. These systems challenge the power of vehicle dynamics tools in two ways: On the one hand the number and complexity of topologies of drive-trains is almost combinatorial due to the large number of involved drives, clutches, or gears. On the other hand particularly hybrid vehicles represent an almost classical multi-domain system (electronics,

hydraulics, combustion, chemistry, mechanics) with a dramatically rich dynamics in the interaction between the various technical components.

Modelica forms a perfect basis for the modeling of multi-domain systems especially automotive systems [4]. The benefits of its general equation-based formalism has been shown by numerous publications in the fields of vehicle dynamics, climate control, drivetrain modeling, combustion engines, and hybrid powertrains, see e. g. [5–8].

The increasing complexity in automotive system development especially for hybrid vehicles demands for versatile modeling tools, realistic and reproducible virtual testing as well as seamless test rig integration. An ideal software for modeling and simulation of all vehicle-related scenarios off-line and in real-time would combine these aspects. Augmenting the comprehensive CarMaker VVE suite by the multi-physics capabilities of Modelica is a first step towards this direction.

2 Augmenting CarMaker by Functional Mock-Up Units

Extending CarMaker

The generality of the CarMaker VVE allows for the modifications of the vehicle models in an almost arbitrary number of ways. Generic vehicle components such as powertrain, steering, tires, brake system as well as the complete propulsion system might be configured, as shown in Fig. 2. For several reasons cus-

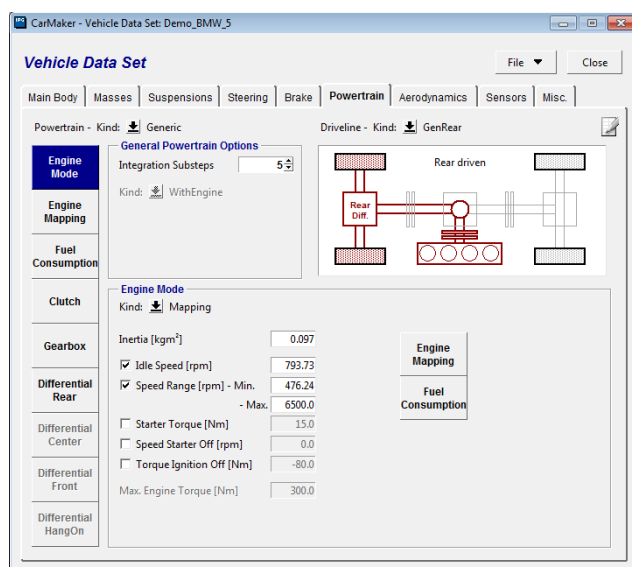


Figure 2: CarMaker interface for vehicle model configuration.

tom models serve as a replacement of the predefined modules in the generic vehicle model or might even replace the vehicle model as a whole: (i) All topologies of hybrid vehicles might not be covered even by an advanced tool specialized in vehicle dynamics. (ii) These multi-physics systems demand for general modeling formalisms and simulation techniques such as Modelica-based approaches. (iii) Many engineering companies and car manufacturers are particularly relying on Modelica during development of components and control units, e.g. [6].

Replacing single vehicle components by so-called custom code is easily done by dynamically linking executable code and registering variables, parameters, and their dimensions and types within the VVE. The requirements when incorporating external code are manifold:

1. Real-time compliance: The code may not rely on specific processing hardware since CarMaker vehicle models should run on arbitrary real-time platforms.
2. Small run-time overhead and memory footprint.
3. Since CarMaker runs its own numerical integration schemes the code of external dynamical models must expose input and output variables. These signals can be physically meaningful, e.g. tire forces, wheel speeds, etc..
4. Implementation details might not be exposed for intellectual property reasons.
5. Standardized, automatic and intuitive inclusion of the external code.

In general, almost all of these requirements can be met by the usage of black-box models, except for the last item. On the other hand these shift problems to different classes: Lack of numerical robustness and/or computational performance.

Functional Mock-Up Interface

The FMI addresses almost all requirements discussed in the previous section. It defines an open standard interface to be implemented by an executable model called Functional Mock-up Unit (FMU). The FMI functions are used (called) by a simulator to create one or more instances of the FMU, and to run these models, typically together with other models. An FMU may either be self-integrating (FMU for co-simulation, FMU-CS)

or require the simulator to perform numerical integration (FMU for model exchange, FMU-ME) [2]. The artifacts which can be packaged in an FMU-file are C-source code, executable code compiled for one or more platforms, and XML descriptions of variables, parameters, and general solver and model properties.

Interfacing Modelica Models to CarMaker

Choosing the FMI as a basis of the interface is beneficial for several reasons since the exposed C-interface is standardized and XML model description [9] included in any FMU contains excellent structural information about the model. The prerequisite is a modeling tool, e.g. Dymola or SimulationX, which generates this description while exporting the Modelica model as C-code. Alternatively this FMU can be hand-coded or generated from any other modeling tool that supports FMI, e.g. a multibody-package.

When interfacing Modelica models to CarMaker one must consider two key aspects: (i) The executable model or code must represent the modeled dynamical behavior without any restrictions in order to maintain generality and to avoid context dependent model semantics. (ii) The resulting model must fulfill requirements essential for real-time applications, most prominently a fixed integration stepsize.

The Interface consists mainly of two parts. The first part governs static type-checking of the input and output variables and parameters of FMU to assure a smooth setup of the solver, generation of user interfaces for parameter input, and automatic extension of the so-called data dictionary for tracing relevant variables during simulation. Finally the executable model is instantiated and connected to the CarMaker solver. Depending on the specific application each type of FMU can be chosen.

Even if the mean computational performance of two coupled simulators is sufficient it can be difficult to assure definite turn-around times, an aspect critical in e.g. HIL systems. CarMaker provides a simulation engine for robust fixed stepsize time integration so including an FMU-ME is a safe solution for simulating the combined vehicle model in the following cases: When either the model complexity is reasonably low, the eigenvalues of the system are located in the stable region of the integration scheme and no strong non-linearities are in the imported model. For FMU generated from very complex or black-box models the option FMU-CS is advantageous, because the FMU contains the appropriate numerical integration scheme. The interface in combination with CarMaker

serves as the master algorithm in the co-simulation of CarMaker and the imported block. This leads to an adaptive oversampling of CarMaker's time grid which might rule out usage on embedded systems or HIL, but leads to superior numerical stability. Compared to the features offered by the full FMU-CS this interface takes advantage only of some features. The setup is the simple one depicted in Fig. 3 with only one single execution engine where the interface is the simulation master which runs sub-system 2 synchronized to the grid the solver of tool 1, i.e. CarMaker.

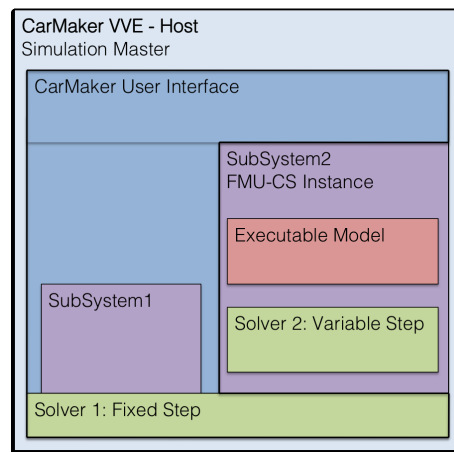


Figure 3: Schematic of run-time architecture.

3 Example Application

The integration of an FMU was investigated for a hybrid truck model based on the Vehicle Dynamics Library [4, 5]. A schematic of the model is depicted in Figure 4. Either the CarMaker standard drive-train or the complete vehicle were replaced within the VVE by an FMU exported from Dymola. In case of the drive-train the driver and environment input signals from the CarMaker model were accelerator pedal, clutch, brake torques, gear number, starter and ignition amongst others. The most prominent model outputs were the wheel speeds.

When using FMU-ME as a replacement the complete vehicle model showed unstable numerical behavior since the fixed solver stepsize did not comply with the dynamics of the new drive-train model. The used stepsize of one millisecond is traditionally used in vehicle dynamics HIL applications. Not even the oversampling feature of CarMaker lead to stable behavior as the constant inputs during the oversampling steps deteriorated the dynamical behaviour. Applying an FMI-CS using an appropriate variable stepsize solver

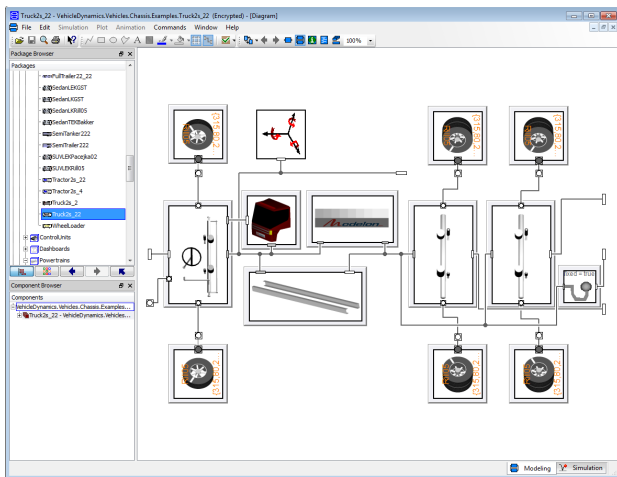


Figure 4: Truck example model in Modelica.

for the drive-train lead to stable numerical simulation within CarMaker. This method allowed the simulation to run in real-time at least in office-mode (soft real-time). Standard techniques to reduce model complexity might help to gain hard real-time capability for HIL applications.

4 Conclusion

This paper presented an interface to augment the CarMaker by functional mock-up interfaces generated from Modelica models. The interface supports both representations of FMI, the one for co-simulation as well as the one for model exchange, exploiting the specific advantages of each approach. The first leads to numerically robust total simulation models where the CarMaker simulation engine is the master algorithm controlling the FMU model containing its own numerical integration scheme and time grid just synchronized on the grid imposed by CarMaker. The latter leads to computationally efficient executable models ideally suited for real-time applications. In the case where the modelled dynamics does not fit to the fixed stepsize prescribed by CarMaker integration scheme this might lead to poor numerical robustness as has been shown by a simple example application.

The interface is not restricted to executable models stemming from Modelica-based tools. FMUs can be generated from dynamic systems by any modeling and simulation tool which supports the FMI standard or can even be hand-coded. The presented work supports a fraction of the features of the current FMI specification V. 1.0. Possible directions are manifold, e. g. network communication between multiple FMUs in a true co-simulation environment e. g. enabled by Sil-

ver [10], parallelization within CarMaker, or the treatment of CarMaker itself as an FMU.

Acknowledgements

The authors would like to thank Christian Schyr, IPG GmbH, and Johan Andreasson, Modelon AB, for valuable comments.

References

- [1] Christian Schyr, B. Bernius, U. Haag, and M. Kircher. Engine-in-the-Loop – Efficient Use of Simulation on the Engine Test Rig. In *IPG Technology Conference apply & innovate 2010*, Karlsruhe, Germany, 2010.
- [2] MODELISAR consortium. Functional Mock-up Interface for Model Exchange, V. 1.0. <http://www.modelisar.org>, 2010.
- [3] *CarMaker User’s Guide Version 3.0*. IPG Automotive GmbH, Karlsruhe, Germany, 2009.
- [4] Johan Andreasson. The Vehicle Dynamics Library: New Concepts and New Fields of Application. In *Proceedings of the 8th International Modelica Conference, Dresden, Germany, 20–22 March 2011*.
- [5] Johan Andreasson and Mats Jonasson. Vehicle Model for Limit Handling: Implementation and Validation. In *Proceedings of the 6th International Modelica Conference, Bielefeld, Germany, 3–4 March 2008*, pages 327–332.
- [6] Henrik Wigermo, Johannes von Grundherr, and Thomas Christ. Implementation of a Modelica Online Optimization for an Operating Strategy of a Hybrid Powertrain. In *Proceedings of the 6th International Modelica Conference, Bielefeld, Germany, 3–4 March 2008*, pages 487–492.
- [7] Sanaz Karim and Hubertus Tummescheit. Controller Development for an Automotive Ac-system using R744 as Refrigerant. In *Proceedings of the 6th International Modelica Conference, Bielefeld, Germany, 3–4 March 2008*, pages 477–485.
- [8] Christian Schyr and Kurt Gschweidl. Model-based Development and Calibration of Hybrid Powertrains. SAE Technical Paper, 2007.

- [9] Francesco Casella, Filippo Donida, and Johan Åkesson. An XML Representation of DAE Systems obtained from Modelica Models. In *Proceedings of the 7th International Modelica Conference, Como, Italy, 20–22 September 2009*, pages 243–250.
- [10] Silver. Qtronic GmbH, Berlin, Germany. www.qtronic.de.

Minimal Equation Sets for Output Computation in Object-Oriented Models

Vincenzo Manzoni Francesco Casella

Dipartimento di Elettronica e Informazione, Politecnico di Milano

Piazza Leonardo da Vinci 32, 20133 Milano, Italy

{manzoni, casella}@elet.polimi.it

Abstract

Object-oriented models of complex physical systems can have a very large number of equations and variables. For some applications, only a few output variables of the model are of actual interest. This paper presents an application of the well-known Tarjan's algorithm, that allows to automatically select the minimal set of equations and variables required to compute the time histories of selected outputs of a given model. The application of the algorithm to a simple test case is illustrated in the paper.

Keywords: Structural analysis, BLT, reduced-order models

1 Introduction and motivation

Object-oriented models of complex physical systems, that can be handled by state-of-the-art tools, can easily count tens or even hundreds of thousands of equations and variables. It is often the case that many of these variables are computed in order to provide the end user with additional information about the behaviour of the system (e.g., 3D visualization of multibody systems) but are not always needed for some applications of the model. In fact, in some cases, only a very few variables are of actual interest for the user.

For example, consider the full dynamic model of a vehicle, built with the MultiBody library of the Modelica Standard Library. During the development of the model, it is of course interesting to visualize all the details in the motion of the suspension system, also for the sake of verifying the correctness of the model. Consider now two applications of this model: a real-time simulator of the car for pilot training, where the simulator cockpit is moved by actuators in order to somehow reproduce the accelerations that the pilot would feel on the real vehicle, and the design of an active suspension system for the same vehicle.

In the first case, the only data which are actually needed at each time step are the orientation and acceleration of the chassis, in order to compute the simulator cockpit motion, and the position and orientation of the windshield, in order to reconstruct a proper 3D view of the outer environment. In the second case, one may run a large number of simulations with different values of some controller parameters, and evaluate some comfort index based on the vertical acceleration of the pilot seat, which is then the only interesting output of the simulation code. In both cases it is important to avoid computing any variable which is not necessary to compute the required outputs, in order to make the time required for the simulation of each time step as short as possible. This might be essential to stay within the sampling time of the real-time simulator, or to avoid an excessively lengthy simulation session in the second case.

A partial solution to this problem is provided by the Modelica language, that allows to define conditional components. One can then include all auxiliary computations (e.g., for visualization) in such components and turn them off by boolean parameters when not needed. This approach has been used extensively in the MultiBody library. A major drawback of this approach is that it requires a significant additional design effort by the library developer; furthermore, it doesn't guarantee that the minimum number of equations which are necessary for the computation of the required outputs is actually selected for the simulation code generation.

The goal of this paper is then to describe an algorithm, based on the well-known strongly connected algorithm by Tarjan, that automatically selects the minimum number of equations and variables in a model which are required for the computation of the time histories of selected output variables. The paper is structured as follows: Tarjan's algorithm is reviewed in Section 2 and applied to the equation selection prob-

lem in Section 3. Section 4 discusses equation selection in the context of dynamic models, while in Section 5 the algorithm is illustrated with reference to a simple case study. Section 6 gives some concluding remarks.

2 Structural analysis by Tarjan's algorithm

Tarjan's algorithm [6] is a well-known graph-theoretical algorithm; its main purpose is to find the *strongly connected components* of a graph.

A graph is an ordered couple $G = (V, E)$, where V is the *vertex set* (or *node set*) and E is the *edge set* (or *arc set*), such that elements in E are couples of elements in V . If the order of the elements in the couple is important, the graph is called *directed graph* (or *digraph*), *undirected graph* otherwise. In particular, a *bipartite graph* (or *bigraph*) is a graph whose vertices can be divided into two disjoint sets V_1 and V_2 , such that every edge connects a vertex in V_1 to one in V_2 .

A directed graph is said to be strongly connected if there is a path from v to w for each couple $(v, w) \in V$. In particular, this means that for each $(v, w) \in V$, a path from v to w exists, as well as a path from w to v . The strongly connected components of a directed graph are its maximal strongly connected subgraphs. For further details, see [2].

Even though different algorithms have been proposed in the literature to compute the strongly connected components – such as Kosaraju's algorithm [5] or the Cheriyan-Mehlhorn algorithm [1] – Tarjan's algorithm is the most known and the one which performs better most of the time.

The algorithm is described in detail in [6]. The basic idea is to perform a depth-first search starting from a start node. The strongly connected components form the subtrees of the search tree; their roots are the roots of the strongly connected components. The complexity of the algorithm is $O(|V| + |E|)$.

For the sake of the presentation, the algorithm is now described by means of examples, shown in Figure 1 and Figure 2. The left hand side of the figures is devoted to show the graph, the right hand side instead shows the stack at each step of the algorithm.

In the first example, the algorithm starts from node 1. In the first three steps, the stack simply records the growing path $1 \rightarrow 2 \rightarrow 3$. At step 3 we find an edge connecting the node at the top of the stack (node 3) to one lower down (node 2). Since we know that there is a path between 2 and 3, this tell us that $(2,3)$ lies on a closed path. This is recorded by putting a frame

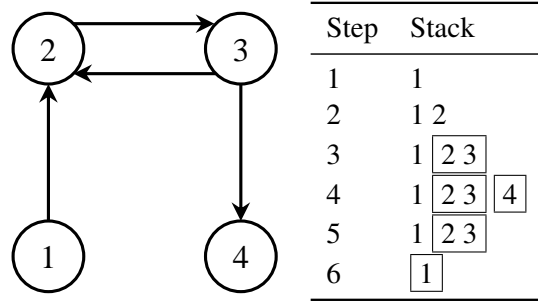


Figure 1: An example of Tarjan's algorithm.

around the nodes which belong to the closed path. We now look for unsearched edges at node 3 and we find that 4 is connected to it. There are no more edges from node 4, which also does not have any link to lower nodes. Therefore, node 4 is labeled as a trivial strongly connected component and removed from the stack. Similarly, there are no more edges from the node 3 and from the node 2; the strong component is removed from the stack. The trivial strong component 1 follows. Summing up, the strong components found in this digraph are 4, 2 3 and 1.

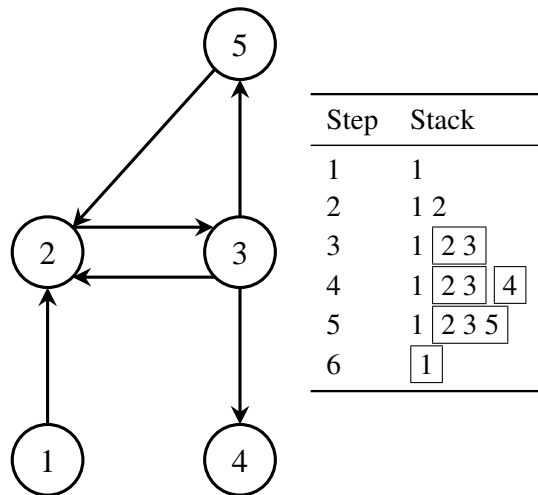


Figure 2: Another example of Tarjan's algorithm.

The second examples shows a more general case. The algorithm starts again from node 1. The first four steps are the same as the previous example. At step 5, node 5 is added to the stack because of the edge $(3,5)$. The edge 5 has a link to a lower node (node 2) which belongs to the strong component. Therefore, node 5 is added to the strong component. Finally, the strong components 1 follows. Summing up, the identified strong components are 4, 2 3 5, and 1.

Step	Stack
1	5
2	5 2
3	5 2 3
4	5 2 3 4
5	5 2 3

Table 1: The stack of the Tarjan’s algorithm applied to the second example, starting from node 5.

Suppose now to apply Tarjan’s algorithm on the same graph as in Figure 2, but starting from node 5 instead. Table 1 shows the stack at each step.

Even if the strong components of the graph are obviously the same, Tarjan’s algorithm finds only two of them (i.e. 5 2 3 and 4). In fact, there are no edges that go from a node to node 1, which has only an outgoing edge.

The algorithm can be easily extended to handle this condition, e.g., by restarting from any node not yet considered. However, this situation can be also exploited for other purposes.

For instance, let’s suppose that the graph in Figure 2 represents dependencies of objects among each other, where nodes model the objects and each arc (v, w) the relation “ v needs w to be evaluated first”. It is then worth noting that only the strong components dependent on node 5 are computed.

3 Minimal equation set selection

One application of Tarjan’s strongly connected components algorithm is the computation of the *Block Lower Triangular* (BLT) form of an incidence matrix. A Block Lower Triangular matrix is a square matrix such that non-zero square blocks are present on the main diagonal, while all blocks above the diagonal are all zeros. An example of BLT matrix is shown in (1), where all matrices $A_{j,j}$ are square.

$$\begin{pmatrix} A_{1,1} & 0 & \cdots & 0 \\ A_{2,1} & A_{2,2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ A_{n,1} & A_{n,2} & \cdots & A_{n,n} \end{pmatrix} \quad (1)$$

This forms allows the corresponding set of equations to be solved as a sequence of subproblems; this is particularly convenient for sparse matrices, which are very common in Object-Oriented models.

The incidence matrix of a system of equations can also be represented by a graph. The application of an

algorithm to compute the strongly connected components of the graph – like Tarjan’s algorithm – allows to determine the BLT transformation. In this section, we show how to apply Tarjan’s algorithm for computing both the BLT transformation and the minimal set of equations and variables according to the selected output variables.

In general, an *incidence matrix* is a matrix which shows the relation between classes X and Y of objects. The size of the incidence matrix is $n \times m$, where the number of rows n is the cardinality of the class X and the number of columns m is the cardinality of the class Y . The matrix element (i, j) is 1 if the object i belonging to the class X and the object j belonging to the class Y are in relationship (or incident), 0 otherwise.

When the incidence matrix is applied to systems of equations, the X class represent the equations and the Y class the variables from which they depend. Only square systems are considered in this paper, i.e., $n = m$.

Consider a generic system of equations S , represented in residual implicit form:

$$S: \begin{cases} f_1(z_1, z_2, \dots, z_n) = 0 & e_1 \\ f_2(z_1, z_2, \dots, z_n) = 0 & e_2 \\ \vdots & \\ f_n(z_1, z_2, \dots, z_n) = 0 & e_n \end{cases}, \quad (2)$$

where e_i ($i = 1, 2, \dots, n$) represent the equations and z_j ($j = 1, 2, \dots, n$) represent the variables of the system. For each equation e_i , the function $f_i(\cdot)$ determines the dependency between the equation and its variables.

The incidence matrix (3) is the structural representation of system (2)

$$\begin{matrix} & z_1 & z_2 & \cdots & z_m \\ \begin{matrix} e_1 \\ e_2 \\ \vdots \\ e_n \end{matrix} & \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,m} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,m} \end{pmatrix} \end{matrix}. \quad (3)$$

The element $a_{i,j}$ is 1 if the residual of equation e_i depends on the value of z_j , 0 otherwise.

An incidence matrix associated to a system of equations can be represented by a bipartite graph $G = (V_1 \cup V_2, E)$, where V_1 is the vertex set which contains the equations (i.e. the rows), V_2 is the vertex set which contains the variables (i.e. the columns) and there exists an arc $(v_i, v_j) \in E$ if the entry (i, j) of the incidence matrix is 1. Such graph is called *equations-variables bipartite graph*, or E-V graph in short.

First, a row permutation of the incidence matrix is computed, such that the value of each entry on the

main diagonal is 1. This has been proven to be equivalent to finding a transversal of the equations-variables graph. A graph transversal is a subset of the edges such that each node belongs only to one arc. This can be computed by using one of the many algorithm in the literature which solve the matching problem, e.g., the Push-relabel algorithm [4] or the Edmonds-Karp algorithm [3].

At the end of the procedure, the incidence matrix will look like:

$$\begin{pmatrix} 1 & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & 1 & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & a_{n-1,n} \\ a_{n,1} & a_{n,2} & \cdots & 1 \end{pmatrix} \quad (4)$$

Finally, Tarjan's algorithm is applied on the graph associated to the diagonal incidence matrix. As we have seen on Section 2, a single run of the algorithm returns different strong components according to the starting node. In particular, if the graph is a dependency graph, only the strong components which depend on the starting node are computed. Therefore, if an output variable is chosen as a starting node, the BLT transformation will only contain the equations and the variables which depend on it.

In order to specify to that one is interested in more than one output variable, a new node s (*source node*) is added to the graph. The node s is then connected by means of outgoing edges to the nodes which represent the output variables. Tarjan's algorithm will then use node s as the starting node. At the equation level, this corresponds to adding to the problem a dummy output variable s and a dummy equation relating s to the required outputs: $s = f_s(y_1, \dots, y_h)$.

4 Application to dynamic models

In the context of object-oriented modelling, continuous-time systems are represented by means of differential-algebraic equations. After flattening and index reduction, the system is described by $n + m$ differential equations:

$$\begin{cases} f_1(x_1, \dots, x_n, \dot{x}_1, \dots, \dot{x}_n, y_1, \dots, y_m) = 0 \\ f_2(x_1, \dots, x_n, \dot{x}_1, \dots, \dot{x}_n, y_1, \dots, y_m) = 0 \\ \vdots \\ f_{n+m}(x_1, \dots, x_n, \dot{x}_1, \dots, \dot{x}_n, y_1, \dots, y_m) = 0 \end{cases}, \quad (5)$$

where x_i ($i = 1, \dots, n$) are the state variables and y_j ($j = 1, \dots, m$) are the algebraic variables. If the states x_i are known at a certain time instant, these equations

can be solved to compute the derivatives and the algebraic variables at the same time. However, for the sake of the equation selection, what really matters is which equations are strictly necessary to compute the *trajectories during time* of the selected output variables, not only at a given initial time t_0 , but for an entire interval $t_0 \leq t \leq t_f$. Therefore, it is necessary to consider the implicit relationship between each variable x_i and its derivative \dot{x}_i , since the latter is uniquely determined once the time history of the former is known.

The incidence matrix for the equation selection algorithm can therefore be set up as follows: the set of variables z_i is given by the state variables x_i , by their derivatives \dot{x}_i , and by the algebraic variables y_j ; the set of equations e_i is given by the set (5), augmented by n dummy equations, each relating a state variable x_i with its derivative \dot{x}_i . The algorithm described in Section 3 is then applied to the resulting E-V graph.

If the object-oriented model is hybrid, i.e., it also contains discrete variables and discrete equations that are active only at events (inside when-clauses in Modelica), the above-described procedure can be suitably extended. In this case, the set of variables z_i should also include the discrete state variables q_h ($h = 1, \dots, u$), their previous values $pre(q_h)$, and all other discrete variables r_k ($k = 1, \dots, v$), while the set of equations should also contain all the $u + v$ discrete equations contained inside the when clauses, as well as u dummy equations relating each discrete state variable q_h with its corresponding previous value $pre(q_h)$.

5 Case study

A simple problem is now used to explain how the algorithm works. Consider the continuous-time dynamical model (6). It has 3 state variables (x_1 , x_2 , and x_3) and two algebraic variables (y_1 and y_2).

$$\begin{cases} \dot{x}_1(t) = -x_1(t) \\ \dot{x}_2(t) = x_1(t) - x_2(t) \\ \dot{x}_3(t) = x_1(t) \\ y_1(t) = 3x_2(t) + x_1(t) \\ y_2(t) = 2x_3(t) \end{cases}. \quad (6)$$

For sake of conciseness, hereafter the time dependency is omitted.

The system is written in explicit form. It is apparent that the value of the algebraic variable y_1 does not depend on the value of the state variable x_3 , neither directly nor indirectly. Similarly, the value of the algebraic variable y_2 does not depend on the value of the state variable x_2 .

The system is now rewritten in implicit form (7); e_5 , e_6 and e_7 are the three dummy equations added to represent the implicit relationship between each variable and its derivative.

$$\begin{cases} x_1 + x_1 = 0 & e_0 \\ x_2 - x_1 + x_2 = 0 & e_1 \\ \dot{x}_3 - x_1 = 0 & e_2 \\ y_1 - 3x_2 - x_1 = 0 & e_3 \\ y_2 - 2x_3 = 0 & e_4 \\ x_1 - f(\dot{x}_1) = 0 & e_5 \\ x_2 - g(\dot{x}_2) = 0 & e_6 \\ x_3 - h(\dot{x}_3) = 0 & e_7 \end{cases} \quad (7)$$

The incidence matrix associated to the system (7) is shown in (8).

$$\begin{matrix} & x_1 & x_2 & x_3 & \dot{x}_1 & \dot{x}_2 & \dot{x}_3 & y_1 & y_2 \\ e_0 & \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix} \end{matrix} \quad (8)$$

The first step of the algorithm is to find a permutation of the rows of the matrix (8) such that the resulting matrix is diagonal. As outlined in Section 3, this can be done finding a transversal of the E-V graph. The diagonal matrix is shown in (9).

$$\begin{matrix} & x_1 & x_2 & x_3 & \dot{x}_1 & \dot{x}_2 & \dot{x}_3 & y_1 & y_2 \\ e_0 & \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \end{matrix} \quad (9)$$

The rows and the columns of the diagonal matrix (9) are now relabeled to ease the definition of the digraph. The symbol ${}^j e_i$ indicates that the equation e_i lies in the j -th row of the diagonal matrix. Similarly, ${}^j x_i$ indicates that the variable x_i lies in the j -th column. Matrix (10)

shows the result of the relabeling.

$$\begin{matrix} & {}^0 x_1 & {}^1 x_2 & {}^2 x_3 & {}^3 x_1 & {}^4 x_2 & {}^5 x_3 & {}^6 y_1 & {}^7 y_2 \\ {}^0 e_0 & \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \end{matrix} \quad (10)$$

Assume now that only the value of y_1 is of interest as a system output. Figure 3 shows the graph associated to the matrix (10). The output y_1 is represented by the node with a bold border. Bold arrows connect the output variable to the state variables which have a direct impact on it.

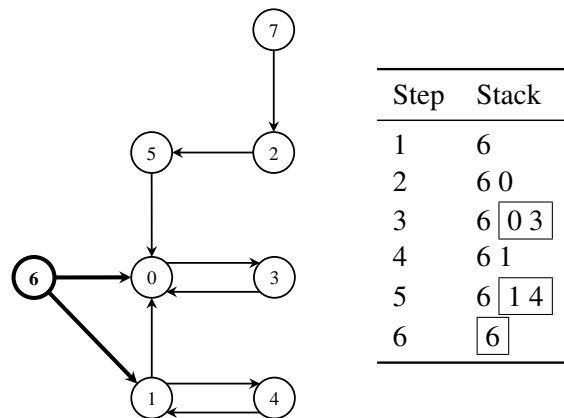


Figure 3: Graph associated to matrix (10).

Tarjan's algorithm is now applied to this graph, starting from node number 6. The algorithm's stack is shown on the right hand side of the same figure. The strong components identified by the algorithm are **0 3**, **1 4** and **6**, which correspond to equations ${}^5 e_0$, ${}^6 e_1$, and e_3 , respectively.

The incidence matrix output of the algorithm is given in (11). The order of the system has been reduced, since the state variable x_3 and its derivative, as well as the algebraic variable y_2 , do not contribute either directly or indirectly to the value of the output variable y_1 . Moreover, the procedure also returns the

incidence matrix in BLT form.

$$\begin{matrix} & \begin{matrix} \dot{x}_1 & x_1 & \dot{x}_2 & x_2 & y_1 \end{matrix} \\ \begin{matrix} e_5 \\ e_0 \\ e_6 \\ e_1 \\ e_3 \end{matrix} & \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 \end{pmatrix} \end{matrix} \quad (11)$$

As noted in Section 3, more than one output variable can be specified. For instance, assume that we are now interested in the time histories of both y_1 and y_2 . The graph is modified by adding the source node s . This node is connected to the interested output variables, in our case to the node 6 (the output variable y_1) and node 7 (the output variable y_2). The source node can also be seen as dummy output variable $y_s(t)$ appended to the system, whose value is a function of the output variables of interest $y_1(t)$ and $y_2(t)$, so that $y_s(t) = f_s(y_1(t), y_2(t))$.

The graph augmented with the source node and the appropriate edges is represented in Figure 4.

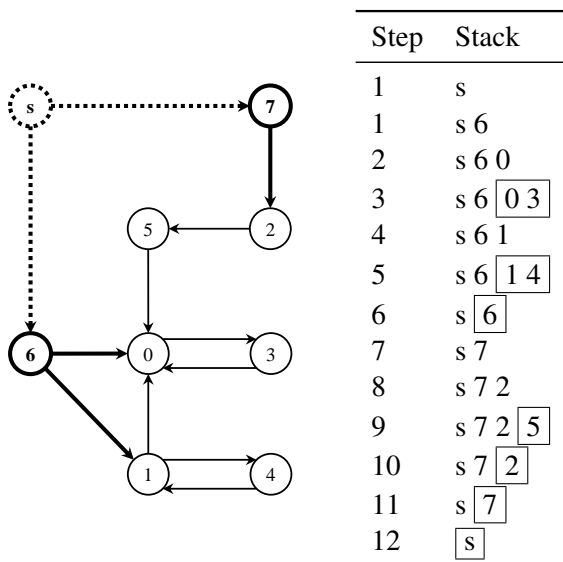


Figure 4: Graph with the source node s .

It is apparent from (7) that in order to compute the value of both the output variables, all the state variables are required. In fact, Tarjan’s algorithm returns the matrix (12) which has the same dimension of the

original one.

$$\begin{matrix} & \begin{matrix} \dot{x}_1 & x_1 & \dot{x}_2 & x_2 & y_1 & \dot{x}_3 & x_3 & y_2 \end{matrix} \\ \begin{matrix} e_5 \\ e_0 \\ e_6 \\ e_1 \\ e_3 \\ e_2 \\ e_7 \\ e_4 \end{matrix} & \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix} \end{matrix} \quad (12)$$

6 Conclusions

This paper presents application of Tarjan’s algorithm to determine the minimal set of variables and equations in an object-oriented model, which are strictly necessary to compute the time histories of selected output variables. The algorithm has been thoroughly illustrated in a simple test case.

This feature can be easily implemented in all Modelica tools and can be very valuable for end-users, when their application does not require to inspect all the model variables and puts a premium on fast simulation performance. In particular, it is planned to implement this feature in the OpenModelica compiler.

Significant applications include real-time code generation, sensitivity or parameter-sweep analysis, and in general all control-oriented applications where the input-output behaviour of the system is of interest.

A particularly nice application could be the case of planar multibody systems, built with the standard Modelica MultiBody library. If only outputs corresponding to the in-plane movement of some points of the system are selected, the procedure illustrated in this paper could allow to remove all the out-of-plane equations of motion, which would then be irrelevant, thus allowing a substantial reduction in the number of equations and state variables of the system.

References

[1] J. Cheriyan and K. Mehlhorn. Algorithms for dense graphs and networks on the random access computer. *Algorithmica*, 15(6):521–549, 1996.

[2] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms*, chapter 22, pages 527–529. MIT Press and McGraw-Hill, second edition, 2001.

- [3] J. Edmonds and R. M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM (JACM)*, 19(2):248–264, 1972.
- [4] A. V. Goldberg and R. E. Tarjan. A new approach to the maximum-flow problem. *Journal of the ACM (JACM)*, 35(4):921–940, 1988.
- [5] R.S. Kosaraju. Unpublished, 1978.
- [6] R. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1:146, 1972.

Optimization example of industrial sewing machines mechanisms

Vlastimil Votrubec Pavel Šidlof
VÚTS, a.s.

U Jezu 4, 461 19 Liberec 4, Czech Republic
vlastimil.votrubec@vuts.cz, pavel.sidlof@vuts.cz

Abstract

This paper deals with the modeling and optimization of mechanical system, focusing on industrial sewing machines in order to reduce the vibrations by balancing. It presents creation of model of mechanisms and their optimization using Modelica language through software MathModelica and Mathematica. Both advantages and drawbacks of this approach are described and an example of optimization solution is shown.

Keywords: optimization; mechanisms; sewing machine

1 Introduction

Our research institution partakes in development of industrial sewing machines focused on reduction of vibrations and noise. These properties significantly influence customer's opinions and lead in this business helps with competition. It is necessary to solve similar problems for other types of machines also.

Vibrations and noise are mainly caused by dynamic forces generated during the movement of mechanisms. To reduce these forces, which grow during the permanent rising of machines performance, different balancing methods are used. The simplest method is balancing by rotatory balancers. More significant reduction of inertial forces is possible to achieve using balancing mechanisms. In our institute industrial used patent was developed in this field.

It is necessary to use appropriate computational software for balance suggestion and optimization. One of the best and universal software is software Mathematica. Industrial sewing machines usually contain complicated system of mechanisms. Their dynamic calculation leads to a large number of equations and parameters. We found that programming of the calculation is difficult and we could easily make mistakes, which were tedious to search. Hence the idea of using software MathModelica to create computational system occurred. Then we can execute

optimization and some other calculations in software Mathematica.

2 Balancing of sewing machines

Sewing machines contain number of mechanisms that cause generation of inertial forces and vibrations. Main of them is mechanism for needle bar motion and thread feed mechanism. Both are placed in the head of the machine. There are crank mechanism that ensures reversible rectilinear movement of needle bar and four bar mechanism for thread feeder motion. Both mechanisms are driven by upper shaft that is connected to lower shaft by transmission belt. Balancing of these two mechanisms could be done by several methods.

2.1 Balancing methods

Common balancing methods of inertial forces balancing use rotatory balancers mounted on crank shaft. This method usually balances only centrifugal forces or transfers first harmonic component of inertial force of needle bar from the direction of needle bar axes to the orthogonal direction. Also this method can't balance higher orders harmonic component of inertial forces, especially in crank mechanism for needle bar motion.

Further, the vibrations are often reduced by increasing of mass and thus increasing of stiffness of sewing machine's head. Mounting in silent-blocks also helps.

Using of balance mechanisms hasn't spread yet, because most of these patented methods are complicated, expensive, require big space inside the machine or they don't balance inertial forces sufficiently. The VÚTS patent, which doesn't have many of these drawbacks, was recently used to balance crank mechanism and partially thread feed mechanism.

2.2 Calculation of mechanisms balancing and its limits

For optimization calculation it is necessary to use convenient software. In these cases software Mathematica was used. During the optimization it is necessary to solve system of equations which includes plenty of parameters. Program written in this way is large and it is easy to make mistakes that are difficult to be found.

Therefore we decided to work with software MathModelica that uses Modelica language and that is compatible with Mathematica. The idea was to create model of mechanisms in model editor using Modelica libraries, especially MultiBody library and then analyze it in Mathematica.

3 Optimization using Modelica language and Mathematica

Modeling of mechanisms has specific requests and many of them aren't implemented in current libraries. Big advantage of this software is the ability to edit elements or even create new element and library.

3.1 Elements editing and creation

Most of the problems that we encounter when working with mechanical systems first require detection of kinematic quantities courses at specific points in bodies and inertial forces (torques). However, elements of the MultiBody library have these quantities defined differently and in smaller number than we need. Some kinematic quantities (velocity and acceleration) are calculated resolved to individual body coordinate system, the total inertial force (torque) of the body isn't defined at all and all variables are primarily calculated only for points of the body where the connection to other elements is modeled. It was necessary to remove these drawbacks so a new element of a rigid body (RigidBody) featuring a link of a mechanism was formed.

In the editor of the new element, new quantities and parameters were defined and relevant equations were set up. Kinematic quantities in points of connection to other elements, which we need to know resolved to global coordinate system, were simply multiplied with appropriate transformation matrix. It is also necessary to work with the courses of quantities of an arbitrary point of the body (such as center of mass) which standard element doesn't provide.

Therefore a radius vector of a general point resolved to coordinate system of the body was set up as a new parameter. Other quantities are then calcu-

lated from this parameter. The position of this point resolved to global coordinate system is calculated as

$$r_{GA} = r_{GL} + S \cdot r_{LA} , \quad (1)$$

where r_{GL} is position vector of the body coordinate system origin, S is transformation matrix, r_{LA} is position vector of a point A resolved to the coordinate system of the body and r_{GA} is searched position of a point A resolved to the global coordinate system.

The velocity and acceleration could be calculated as time derivatives of this formula, but function derivative can't be used, because inserting of derivatives isn't compatible with inner logic of the software and calculation is interrupted. For this reason velocity and acceleration of given point is calculated as

$$v_{GA} = S \cdot v_{GL} + w \times (r_{GA} - r_{LA}) , \quad (2)$$

$$a_{GA} = S \cdot a_{GL} + z \times (r_{GA} - r_{LA}) + w \times (w \times (r_{GA} - r_{LA})) , \quad (3)$$

where w and z are angular velocity and acceleration of the body, indexes GL correspond to the origin of the coordinate system of the body and indexes GA correspond to the searched global velocity and acceleration of the given point.

Inertial force F is defined by multiple of body mass m and its acceleration in the center of mass a_{GT} . The acceleration is calculated according to relations (1, 2, 3), only instead of position vector r_{LA} , the position vector of center of mass r_{LT} is inserted

$$F = m \cdot a_{GT} . \quad (4)$$

In the similar way inertial torque is calculated, only instead of body mass, moment of inertia resolved to the center of mass and angular acceleration of the body are inserted. This method allows us to

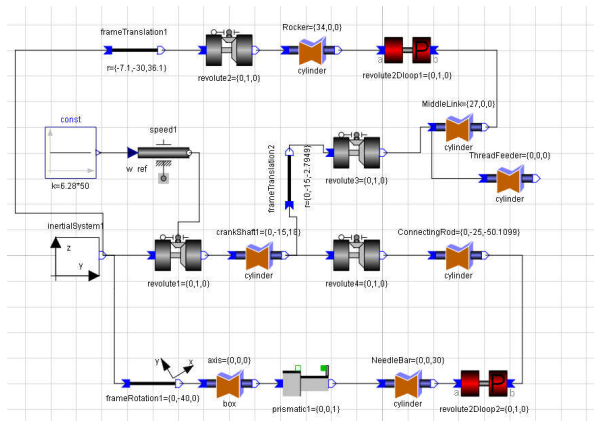


Figure 1: Model of upper mechanism of a sewing machine.

enlarge elements with new parameters and quantities according as necessary.

There can be relatively a lot of mechanisms inside the sewing machine (according to the type of machine). These mechanisms can form a long chain. Already the model of simple mechanism such as crank mechanism or four bar mechanism from fig. 1 includes about ten elements. For easier work and better orientation it is convenient to assemble these simple mechanisms into blocks and form independent elements from them, which are then connected.

The upper mechanism model of a sewing machine from fig. 1 is then simplified to model that has only five elements (fig. 2). Used elements represent global coordinate system, planar crank mechanism and binary revolute pair replacing four bar mechanism of thread feeder, all for suggested constant value of crank shaft angular velocity which is connected to the CrankMech2D element.

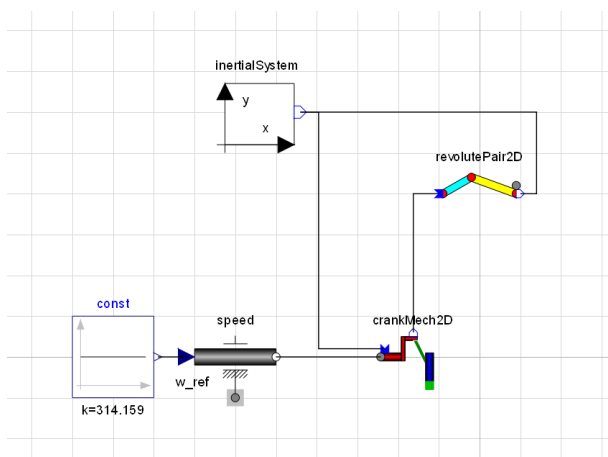


Figure 2: Model of upper mechanisms of sewing machine. Both mechanisms are replaced with single element.

3.2 Optimization

Optimization of chosen parameters of the model is then solved in Mathematica. There is created function for finding local minimum that searches the minimal value for every parameters setting. During every step of finding minimum is solved whole system of equations in MathModelica Simulation Center. There is also possible to use any of other operators that the software offers, such as finding local maximum, difference between two courses, finding of roots of derivatives etc.

Generally the optimization is difficult calculation. During the balancing of mechanisms and machines (reduction of total dynamic forces and torques) it is possible to use specific properties of this problem. Usually mechanisms with periodic movement are

solved and balanced so that optimal dimensions and moments of inertia of several balancing masses are found. Each body is possible to replace by proper chosen point masses if the flexibility is omitted (3 points are enough for plane case). Total inertial force is then sum of inertial forces in these points.

For example it is searching minimum of absolute value of F in vector function

$$F = m_1 \cdot a_1 + m_2 \cdot a_2 + \dots + m_n \cdot a_n. \quad (5)$$

If points are chosen conveniently, then only the mass m_i of points is necessary to find, not their optimum position. Courses of acceleration a_i , which are generally given by very complicated relations, are therefore in particular points stable, they can be calculated before the optimization and don't have to be executed in each optimization step. In order to effective calculation using MathModelica, it is necessary to do some modifications and completion in this software.

3.3 Drawbacks of this approach

There are several difficulties which make optimization complicated. Firstly it is absence of some mechanical elements and joints in MultiBody library. For instance a lot of mechanisms in sewing and other machines consist of cams and other shape bodies which can't be modeled by means of standard elements. Hence, joint between cam and lift and other elements would help much in mechanisms modeling.

Next problem is time of optimization. Elements of mechanism model form system of equations which includes thousands equations. Already simple mechanism such as four bar mechanism has about 3000 equations, 2500 of them are trivial equations. Although the simulation takes a few seconds, it is executed in every step during the optimization. Finding local minimum is a long process with dozens up to hundreds of steps. This finally causes long time of a calculation depends on model difficulty and number of parameters. Each other used parameter for optimization significantly lengthens the time of calculation.

3.4 Possible approaches of next process

One possible solution of the problem with time of optimization was found in solving of whole system of equations independently on Simulation Center which could be much more efficient. MathModelica allows transferring the system of equations to software Mathematica. Mathematica offers many operators for solving equations, the most appropriate operator for this case is function NDSolve that can numerically solve system of differential equations.

This approach also has some problems which don't enable to sufficiently solve equations for this time.

3.5 Optimization example

It will be presented optimization example of balancing on upper mechanisms of sewing machine in this item. As was said before, there are two mechanisms driven by upper shaft, crank mechanism and four bar mechanism. Total inertial force is a sum of particular forces of the bodies. The biggest influences on total inertial force have inertial forces generated by movement of crank shaft and needle bar. The courses of unbalanced total inertial force and its components are on the fig. 3, whereas the crank shaft is rotating 3000 RPM counter clockwise.

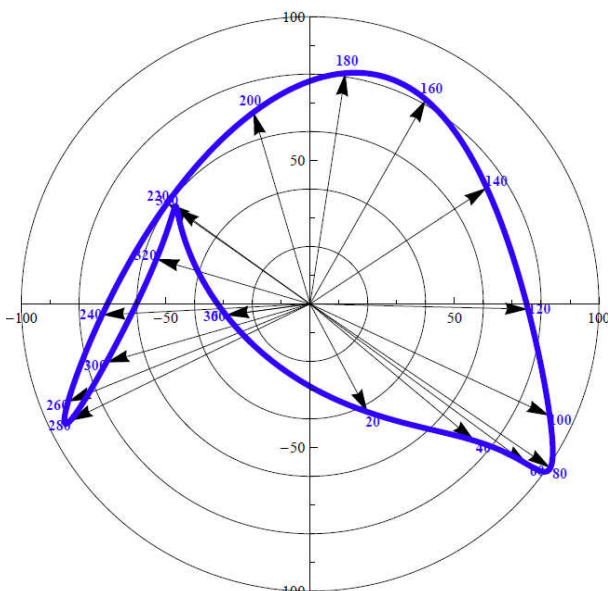


Figure 3: Polar plot of total inertial force of unbalanced upper mechanism. The maximum force is 100,9N.

The polar diagram shows resulting force vector. The arrows present vectors for particular drive shaft angle rotation in degrees.

Generally the reduction of total inertial force can be achieved by using two rotatory balancers on the crank shaft. But in this case the crank shaft is pre-balanced. Position of center of mass is near the axis of rotation so balancing using rotatory balancers can't help much. Significant reduction of the total force is possible to achieve using a balancing mechanism and one rotatory balancer. In our case it consists from connecting rod that is mounted on the crank shaft on the opposite side of the needle bar crank and balancer (fig. 4). The balancer has small stroke but large mass. Also there is one rotatory balancer placed on the crank shaft.

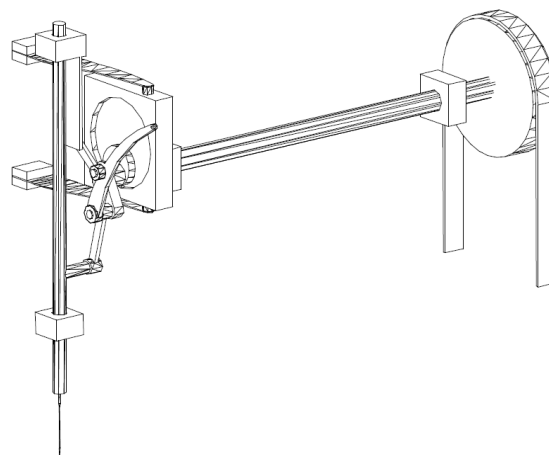


Figure 4: Schema of upper mechanisms of sewing machine. Needle bar mechanism and thread feeder mechanism with additional balancing mechanism.

The dimensions of balancer and balancing mechanism are fixed, so there are two values to optimize, mass of the rotatory balancer and mass of balancer of balancing mechanism. Then in Mathematica is constructed function for calculation of total inertial force, finding the maximum value of that force and the parameters are set. Optimization then implies that the operator for finding local minimum is changing parameters (mass of balancers) until it finds the minimum value.

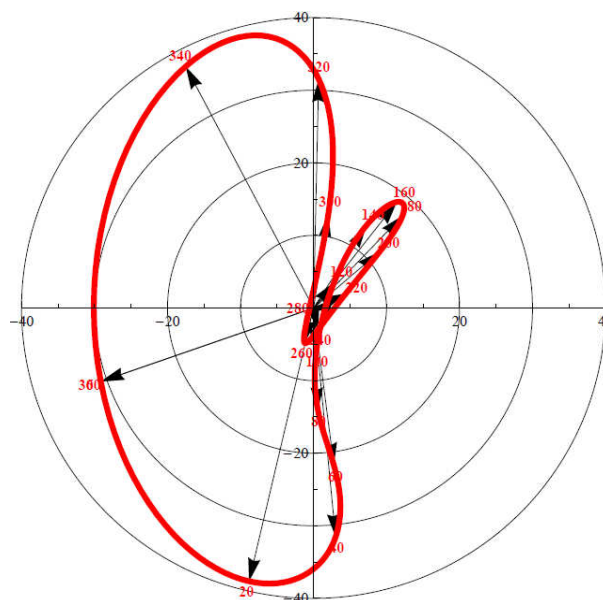


Figure 5: Polar plot of total inertial force of upper mechanisms after the optimization. The maximum force is 38,7N.

Calculation lasts about 60 minutes (3 GHz processor, 1.99 GB RAM), for one parameter it lasts 30 minutes. It is clear that for more complicated model

and more parameters the optimization would take hours. The results are shown on figure 5. The reduction of total inertial force is relatively significant, maximum of total force descended about 60%.

4 Conclusion

Mentioned approach allows modeling of system of mechanisms, executing kinematic and dynamic analysis and optimizing parameters to reduce inertial forces and vibrations. This approach is clear and it reduced mistakes in comparison with other software and methods. On the optimization example of balancers is shown how it is possible to achieve fast and precise results.

On the other hand there is still large area for improving current methods and means in optimizations. For example mechanical elements that corresponds to real links of mechanisms and design problems, possibility to enter more mass parameters etc.

Automatic Generation of Graphical User Interfaces for Simulation of Modelica Models

Clemens Schlegel
Schlegel Simulation GmbH
Meichelbeckstr. 8b
D-85356 Freising
cs@schlegel-simulation.de

Reinhard Finsterwalder
Universität der Bundeswehr
München
D-85577 Neubiberg
reinhard.fensterwalder@unibw.de

Abstract

For a certain class of applications simulation models are developed and then rolled out for standalone usage without the tool with which they have been developed. The user is intended to perform simulation runs, to inspect results, to change selected parameters within given bounds, but not to inspect or even change the model itself.

The reasons for such a usage scenario are manifold: The simulation is intended to be used as a black-box tool by non simulation specialists, a component vendor (electric drives, pneumatic or hydraulic components, etc.) likes to demonstrate the performance of his components in the context of a simulation or the model developer may hide model details.

If a model development tool includes code generation the model specific simulator can be setup fully automatic. However, a GUI (graphical user interface) for such a simulator must be developed manually. We developed a tool which automatically generates a simulator GUI from a Modelica model and data definition.

Keywords: graphical user interface generation; Modelica parser; standalone simulator

1 Introduction

The core functionality of a model-specific simulator is to run a simulation experiment, to inspect trajectories and / or scalar results and to change, store and retrieve parameters. The computational part of such a simulator may be set up easily using a code generating model development tool. However, to our knowledge, there are no tools publically available for automatic generation of the graphical user interface

(GUI) part. On the other hand nearly all information needed to set up a simulator specific GUI is available in the model code. This GUI, which may be used independent of a general simulation environment, can be set up automatically by parsing a Modelica model. We used Dymola [1] for model development and model specific C-code generation and developed an own tool for GUI generation.

1.1 Limitations

Since the generated model specific C-code can't be changed anymore the GUI's usage is restricted to operations which do not require to change variable dimensions or to replace parts of the model. Arrays must have a fixed dimension or be handled via an external C-function with dynamic storage allocation.

1.2 Core requirements

Focusing on the black-box simulation case we define the following desirable functional requirements for the simulator and the GUI. Some of this requirements map directly to Modelica parameter and record declarations.

- The simulation executable is driven by a parameter and simulation control input file and saves computed trajectories in a result file.
- The GUI is generated at run time by parsing Modelica files, the GUI structure is not stored.
- Only model components and output variables declared on the top hierarchy level of the model are available in the GUI. Protected, inherited, modified and replaced declarations are taken into account, no restrictions on the Modelica language apply.

- Only parameter record classes and top hierarchy model classes (including all inherited and redeclared classes) must be available as Modelica code for GUI generation. Thus the model may contain references to confidential libraries without disclosure of the corresponding Modelica code, external function calls and encrypted classes (if allowed by the modeling tool).
- Names and attributes of Modelica parameters and output variables and restrictions on parameters (protected, read only, min/max values) are retained in the GUI.
- Based on parameter record class parameterization drop-down lists for parameter record selection are set up. This means a model provider (or even a GUI user) may later add parameter records which automatically show up in the corresponding selection list without rebuilding the simulator executable. These records may contain any data type including arrays, optionally a whole record can't be modified and / or its content is not visible. This feature facilitates the usage of datasheet libraries without disclosure of data details.
- It must be possible to read data from external files at runtime without fixed file names.
- All model parameters including records, record names and modified records may be stored on and read from file. The tool must check the consistency of the model, the read in parameter file and the parameter record definitions.

1.3 Additional functionality

Apart from the model parameterization and simulation experiment settings some more information is needed for automatic generation of a handy GUI:

- The top model file name and class name
- The name of the simulator executable
- The user may specify predefined trajectory plots
- The user may specify predefined reports containing descriptive text and trajectory final values

All this information is stored in a configuration file.

2 Parsing the Modelica model

For parsing Modelica [2] files the parser generator tool PCCTS [3] has been used. Based on a grammatical description of a formal language, PCCTS generates C++ code of a corresponding parser which we integrated in our application.

2.1 Parsing the model and the parameter input

All Modelica files in a directory structure are parsed starting from the directory where the configuration is stored. The parser builds an abstract syntax tree for the Modelica code of the complete model hierarchy. The parser fully supports inheritance, modifications and redeclarations within the model hierarchy. Only the data needed for setup of the GUI are retained: model components, model outputs, component parameters, and parameter records. All other tokens are skipped, e.g. algorithm clauses, connect statements, equation clauses, arithmetic expressions.

Since only classes in the model hierarchy are parsed, no information is available from model libraries outside of that hierarchy. In order to generate a complete parameter input file, the remaining parameters are found by parsing the default parameter input file of the simulator (for Dymola it may be generated by the simulator itself).

2.2 Consistency check

It must be checked whether the simulation executable has been generated from the parsed model. Therefore it is verified that all parameters of the model are consistent with the parameters of the parameter input file which may be read in to retrieve previous parameter settings.

2.3 GUI setup

By walking through the abstract syntax tree the graphical user interface is built: tabs for components and controls for model parameters are created and drop-down lists are set up for selecting replaceable parameter records. Thus the user can switch records without having to rebuild the simulation executable. This feature is beyond the functionality of most Modelica tools. Since record names are also stored on the parameter file record selection settings (and not only parameter settings) of previous simulation runs can be retrieved. Modified parameter values are marked red in the GUI.

3 The generated simulator GUI

3.1 General description

At runtime the GUI generator needs the following files:

- Configuration file

- Modelica code of parameter record classes and top hierarchy model classes including all inherited and redeclared classes
- Model specific simulator executable

To start the GUI generation the configuration file is read in. After parsing the tool displays a menu and icon bar and three areas: a model and output navigation tree, a multi document view area for model component parameters, trajectory and report display, and a message area (figure 2, underlying Modelica model see figure 1). Concerning model parameters the GUI has the same functionality as a general simulation tool. For each top level model component a tab is displayed which contains all parameter and record selection controls. Parameter name, default value, unit and descriptive text are shown. Clicking the symbol right of a record selection control the contents of the chosen record is shown. If not protected or read-only parameters may be edited within the defined limits. If a parameter contains a file name the contents of that file is displayed by clicking the respective control (figure 3).

3.2 Performing simulation experiments

The complete parameter setup may be stored on or read from file. To retain the user's parameter changes a new parameter input file is generated before starting a simulation run. That parameter file is enriched by the names of the selected data records. The default simulation experiment setup is contained in the parameter file, but may be overridden using the GUIs simulation menu. After a simulation run the simulation log file is shown in the GUI message area.

3.3 Reading data from file

Since arrays must have a fixed dimension (see section 1.1) a model developer will make use of external tables stored on file handled via an external C-function with dynamic storage allocation. If a model development tool does not support change of string parameters without C-code rebuild a workaround has to be implemented to allow reading from different files. To do so the simulation model has to define fixed file names while the corresponding data records contain a parameter with the specific file name. When starting a simulation run, the specific parameter files are copied to the current directory and renamed to the predefined fixed file names.

3.4 Trajectory and report display

The GUI supports the display of multiple diagrams and reports in multiple windows. While multiple y-axes are supported, trajectories with same unit and similar range are automatically displayed with a common y-axis. Numeric values of a selected trajectory are displayed in addition in a list box (figure 4). Predefined reports show up in a separate window of the multi document view area (figure 4). Trajectories from a previous simulation run may be read in from file and displayed for comparison or used for additional reports.

4 Conclusion

In order to facilitate the development of black-box simulators with a fixed model the automatic generation of simulator specific graphical user interfaces is desirable. A practical approach for building the GUI of such a simulator for Modelica models has been presented. Parsing the model and the data definitions a GUI is set up at run time. The main issues are to check the consistency of the model, the parameter file and the parameter record definitions and to map a smart limitation of the parameter space to the GUI in order to ensure that the simulator can be used as a black-box.

Future development will focus on additional consistency checks and user comfort. Pre-parsed model and data definitions may be integrated directly in a binary code format of the GUI, thus avoiding to read in the model at runtime. The documentation embedded in a Modelica model may be made available in the GUI. Simulation log files may be evaluated automatically to detect convergence problems and event chattering.

References

- [1] Dymola Version 7.4. Dassault Systèmes, Lund, Sweden. www.dymola.com.
- [2] Modelica Specification, Version 3.1, May 2009. www.modelica.org/documents.
- [3] Parr, T.J., Language Translation using PCCTS and C++. Automata Publishing Company, San Jose, 1993.

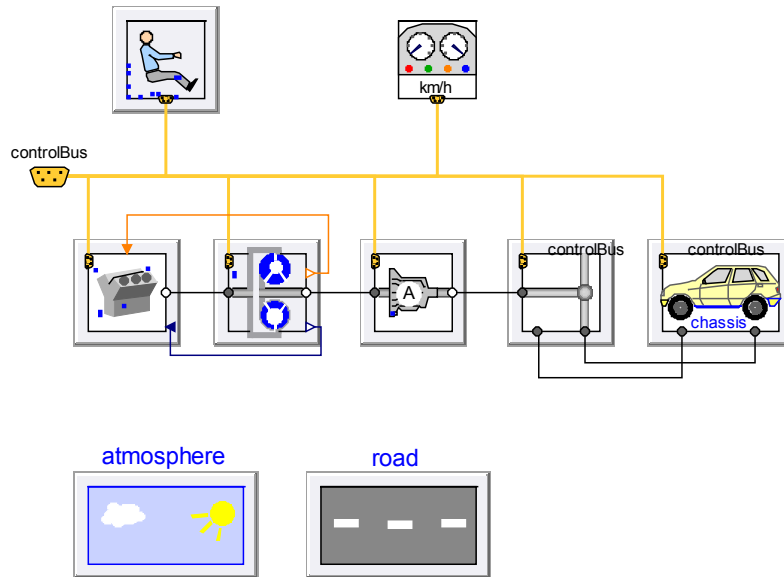


Figure 1: A Modelica simulation model

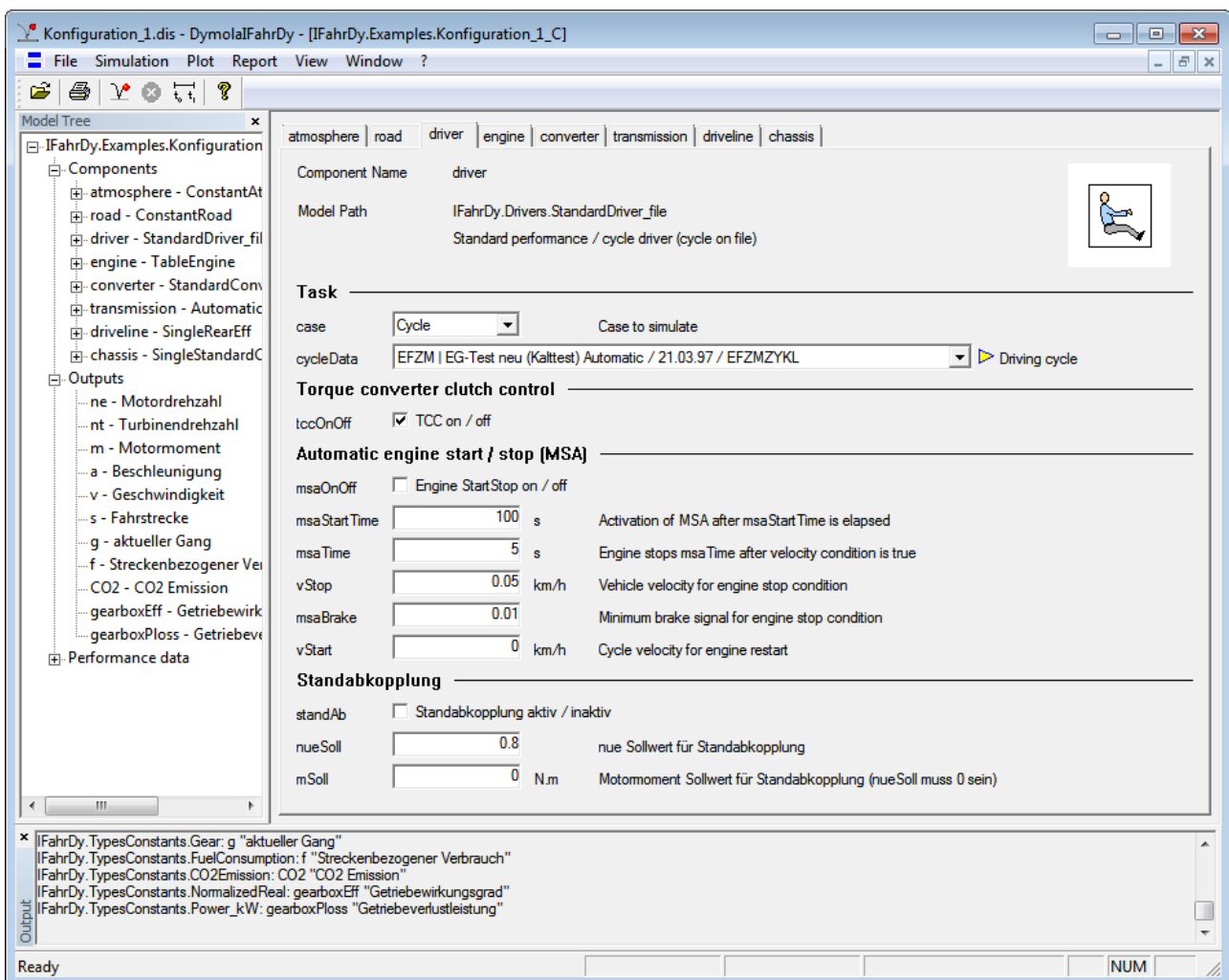


Figure 2: GUI generator main view (model parsed)

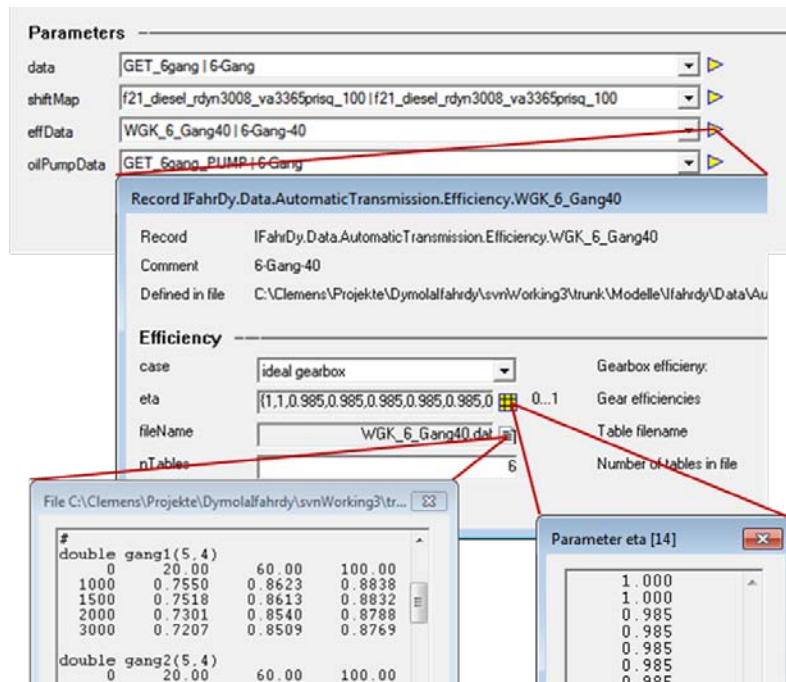


Figure 3: Display of record, file and array contents

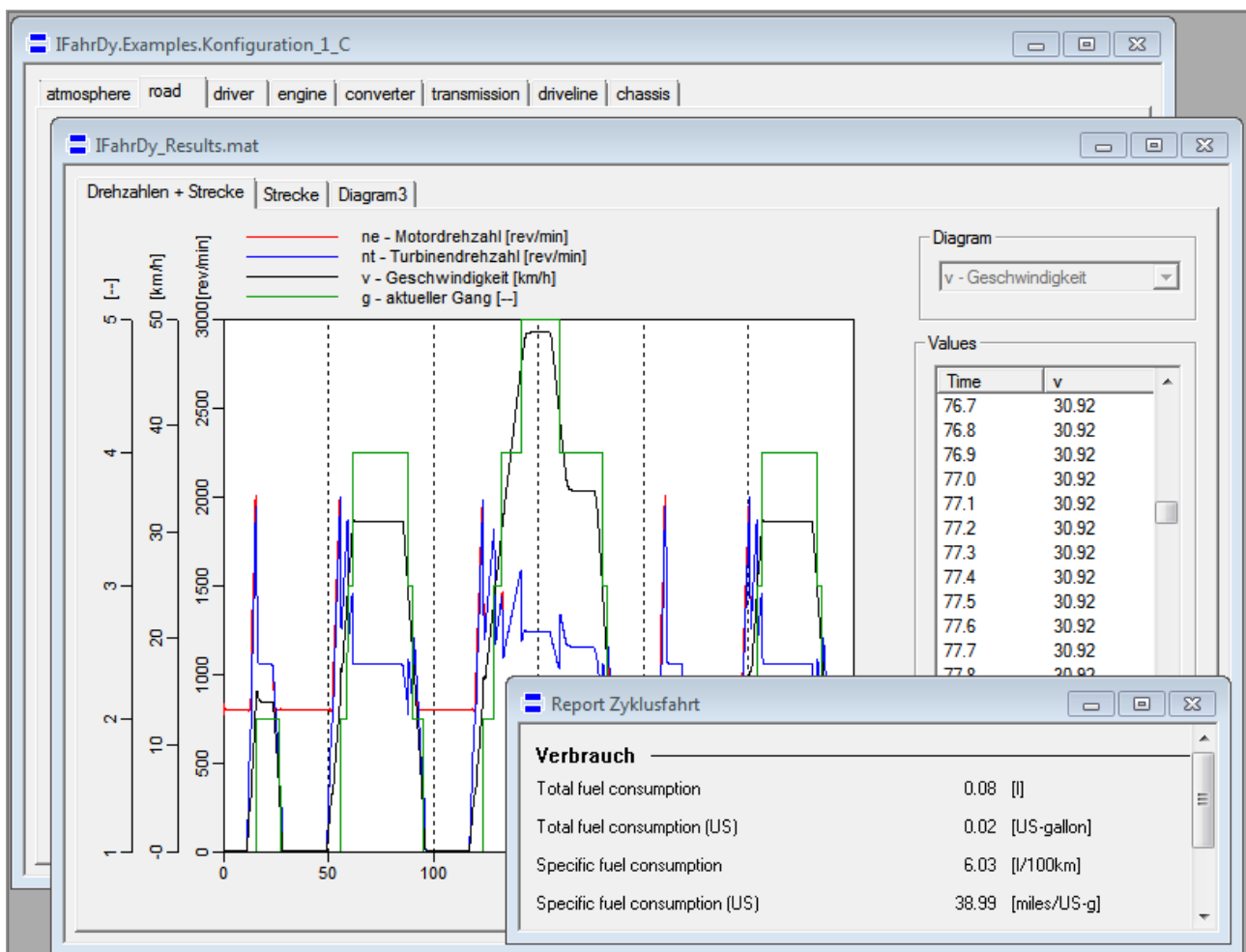


Figure 4: Trajectory and report display

DrControl — An Interactive Course Material for Teaching Control Engineering

Mohsen Torabzadeh-Tari, Martin Sjölund, Adrian Pop, Peter Fritzson

PELAB – Programming Environment Lab, Dept. Computer Science
 Linköping University, SE-581 83 Linköping, Sweden
 {mohsen.torabzadeh-tari, martin.sjolund}@liu.se
 {adrian.pop, peter.fritzson}@liu.se

Abstract

In this paper we present an interactive course material called DrControl for teaching control theory concepts mixed together with exercises and example models in Modelica.

The active electronic notebook, OMNotebook, is the basis for the course material. This can be an alternative or complement compared to the traditional teaching method with lecturing and reading textbooks. Experience shows that using such an electronic book will lead to more engagement from the students. OMNotebook can contain interactive technical computations and text, as well as graphics. Hence it is a suitable tool for teaching, experimentation, simulation, scripting, model documentation, storage, etc.

Keywords: DrControl, DrModelica, modeling, simulation, OMNotebook, teaching, interactive, Control

1 Introduction

In this paper we introduce an electronic interactive course material called DrControl and its use for teaching control theory together with control applications in Modelica [1] [2]. It is developed in and uses the OMNotebook [5] active electronic book software together with OpenModelica for modeling and simulation.

This kind of interactive courses based on electronic books allows experimentation and dynamic simulation as well as execution of computer programs.

Traditional teaching methods with lecturing and reading a textbook are often too passive and does not engage the student. Active notebooks, however, facilitates the learning process, e.g. by running programs and exercises within the book, and mixing lecturing with exercises and with reading in the interactive book.

Electronic notebooks created using OMNotebook can contain program code, text, links, pictures, video,

virtual and scientific visualizations, and makes it possible to integrate teaching material in sciences such as physics, human biology [3], mathematics, computer science, etc.

1.1 Structure of the Paper

Section 2 presents the OMNotebook tool, whereas Section 3 describes the teaching goals and contents of the DrControl electronic book. Section 4 briefly mentions applications in teaching modeling and programming languages, whereas Section 5 presents future work and Section 6 gives the conclusions..

2 OMNotebook – An Active Electronic Notebook

The OpenModelica Notebook editor, OMNotebook, provides an active electronic notebook including an editor. The notebook it is not just a passive textbook or html page, it is active in the sense that models inside the book can be changed and executed.

This functionality allows the usage of interactive hierarchical text documents where the underlying chapters and sections can be represented and edited. OMNotebook supports functionality for Modelica model simulation [1] [2], text, images and interactive linking between those. Furthermore, via the external interface, program in other languages can be evaluated. One example is OMScheme (Section 4.2) for teaching the Scheme programming language.

The hierarchical structure of traditional documents, e.g. books and reports, can also be applied to the notebook which means basically that the book is divided into sections, subsections, paragraphs, etc. This makes the navigation in the book sections easier.

2.1 DrControl

Application of OMNotebook in control theory with the DrControl course material aims at reinforcing the understanding through practical applications with hands-on experience. The students gain insight into the dynamic phenomena of a system. Also, the problem-solving process can be built into the material thus letting the students explore the content at his or hers own convenience.

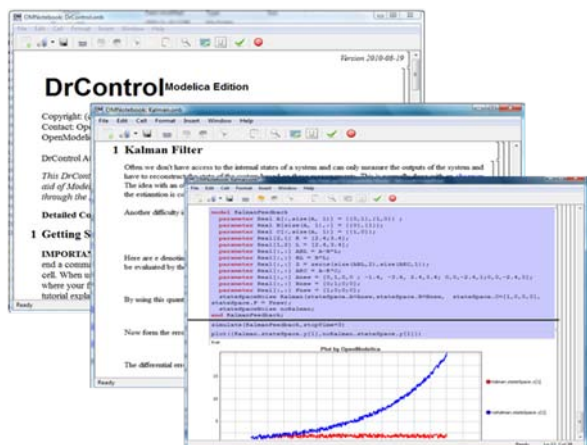


Figure 1. DrControl for teaching control theory concepts.

3 Content and Learning Goals of DrControl

One important factor in modeling and simulation is the availability of the source code, documentation of the source code as well as the result of the simulation in the same document. This is important because the problem solving process is an iterative process that requires modification of the original mathematical model and/or the software implementation and verification of the simulation result against the model.

The front-page of DrControl shown in Figure 2 resembles a linked table of content that can be used as a navigation center. The content list contains topics like:

- Getting started
- The control problem in ordinary life
- Feedback loop, see Section 3.1
- Mathematical modeling, see Section 3.2
- Transfer function, see Section 3.3
- Stability
- Example of controlling a DC-motor
- Feedforward compensation
- State-space form, see Section 3.4

- State observation, see Section 3.5
- Closed loop control system.
- Reconstructed systems, see Section 3.5
- Linear quadratic optimization, see Section 3.6
- Linearization, see Section 3.7

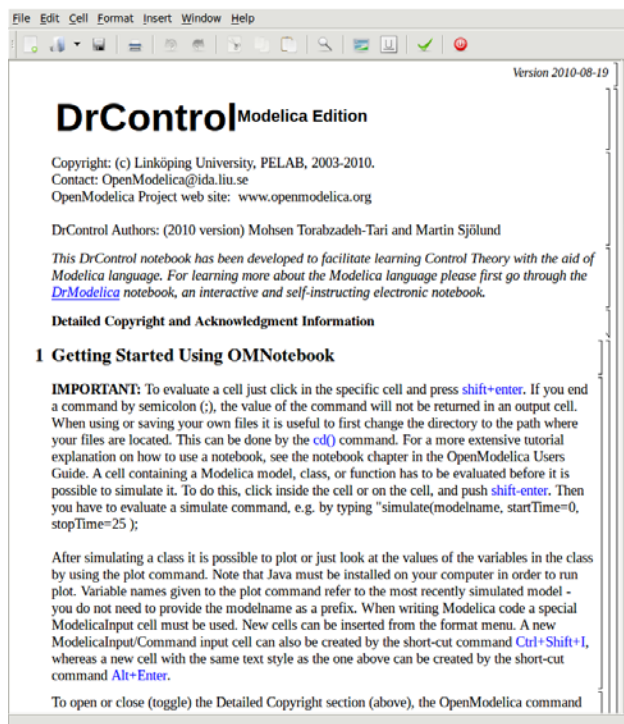


Figure 2. The starting page of the DrControl tutoring system using OMNotebook.

Each entry in this list leads to a new notebook page where either the theory is explained with Modelica examples or an exercise with a solution is provided to certify the background theory, see [7] for more information and down-load of DrControl.

3.1 Feedback Loop

One of the basic concepts of control theory is using feedback loops either for neutralizing the disturbances from the surroundings or a desire for a smoother output.

In Figure 5 a simple car model is illustrated where the car velocity on a road is controlled, first with an open loop control then compared to a closed loop system with a feedback loop. The car has a mass m , velocity y , and aerodynamic coefficient α . The θ is the road slope, which in this case can be regarded as noise.

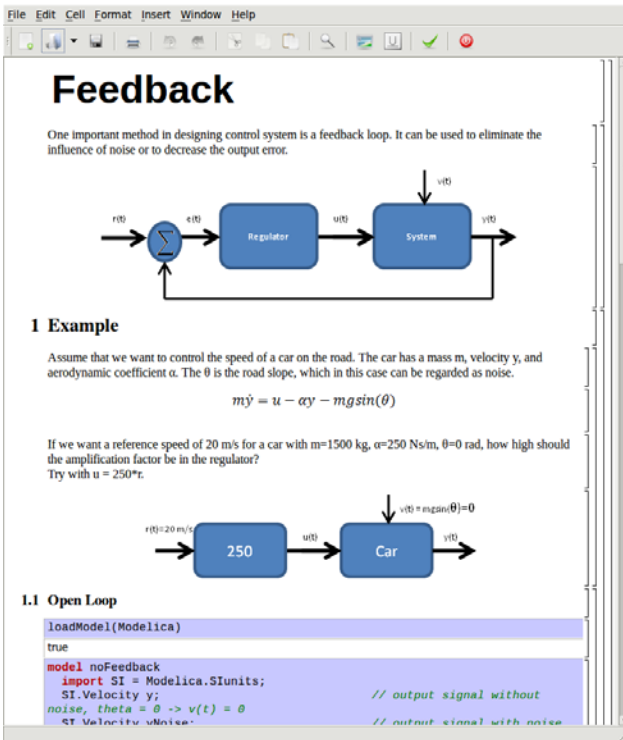


Figure 3. Feedback loop.

Lets look at the Modelica model for the open loop controlled car:

$$m\dot{y} = u - \alpha y - mg\sin(\theta)$$

```
model NoFeedback
import SI = Modelica.SIunits;
SI.Velocity y "No noise";
SI.Velocity yNoise "With noise";
parameter SI.Mass m = 1500;
parameter Real alpha = 200;
parameter SI.ngle theta = 5*3.14/180;
parameter SI.Acceleration g = 9.82;
SI.Force u;
SI.Velocity r = 20 "Reference signal";
equation
m*der(y)=u - alpha*y;
m*der(yNoise)= u - alpha*yNoise -
m*g*sin(theta);
u = 250A*r;
end NoFeedback;
```

By applying a road slope angle different that zero then the car velocity is influenced which can be regarded as noise in this model. The output signal in Figure 3 is stable but an overshoot can be observed compared to the reference signal. Naturally the overshoot is not desired and the student will in the next exercise learn how to get rid of this undesired behavior of the system.

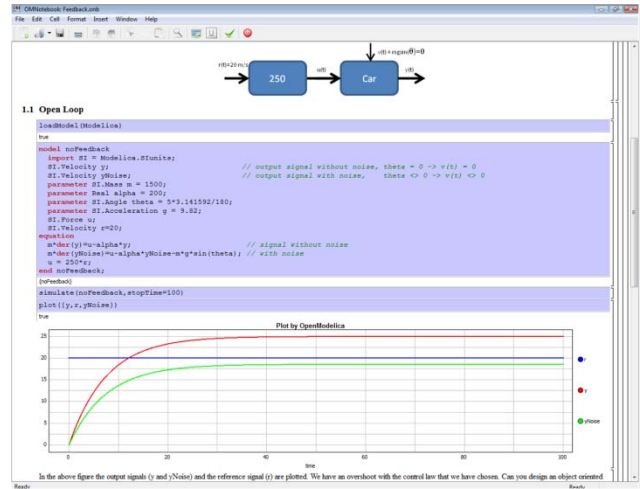


Figure 4. Open loop control example.

The closed car model with a proportional regulator is shown below:

$$u = K * (r - y)$$

```
model WithFeedback
import SI = Modelica.SIunits;
SI.Velocity y "Output, No noise";
SI.Velocity yNoise "Output With noise";
parameter SI.Mass m = 1500;
parameter Real alpha = 250;
parameter SI.Angle theta = 5*3.14/180;
parameter SI.Acceleration g = 9.82;
SI.Force u;
SI.Force uNoise;
SI.Velocity r = 20 "Reference signal";
equation
m*der(y) = u - alpha*y;
m*der(yNoise) = uNoise - alpha*yNoise -
m*g*sin(theta);
u = 5000*(r - y);
uNoise = 5000*(r - yNoise);
end WithFeedback;
```

By using the information about the current level of the output signal and re-tune the regulator the output quantity can be controlled towards the reference signal smoothly and without an overshoot, as shown in Figure 5.

In the above simple example the flat modeling approach was adopted since it was the fastest one to quickly obtain a working model. However, one could use the object oriented approach and encapsulate the car and regulator models in separate classes with the Modelica connector mechanism in between.

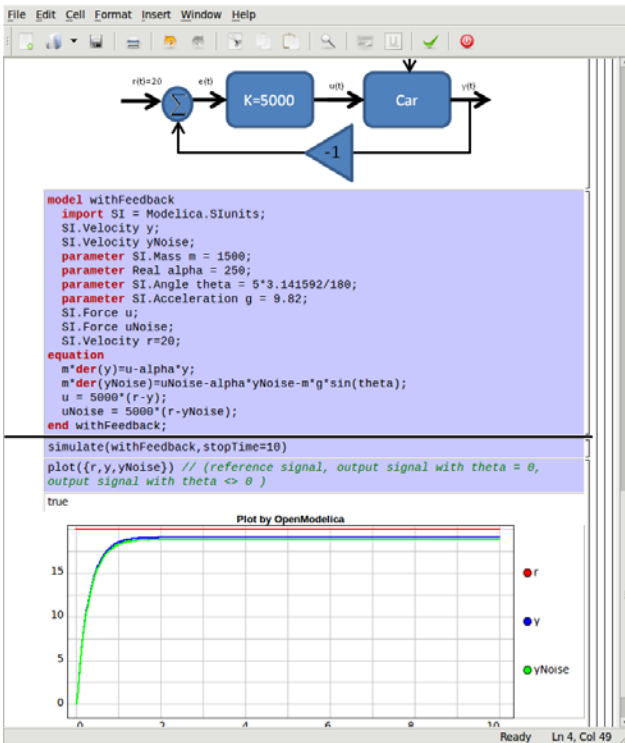


Figure 5. Closed loop control example.

3.2 Mathematical Modeling

In most systems the relation between the inputs and outputs can be described by a linear differential equation. Tearing apart the solution of the differential equation into homogenous and particular parts is an important technique taught to the students in engineering courses, also illustrated in Figure 6.

$$\frac{d^n y}{dt^n} + a_1 \frac{d^{n-1} y}{dt^{n-1}} + \dots + a_n y = b_0 \frac{d^m u}{dt^m} + \dots + b_{m-1} \frac{du}{dt} + b_m u$$

Now let us examine a second order system:

$$\ddot{y} + a_1 \dot{y} + a_2 y = 1$$

```

model NegRoots
  Real y;
  Real der_y;
  parameter Real a1 = 3;
  parameter Real a2 = 2;
equation
  der_y = der(y);
  der(dер_y) + a1*der_y + a2*y = 1;
end NegRoots;
    
```

Choosing different values for a_1 and a_2 leads to different behavior as shown in Figure 7 and Figure 8.

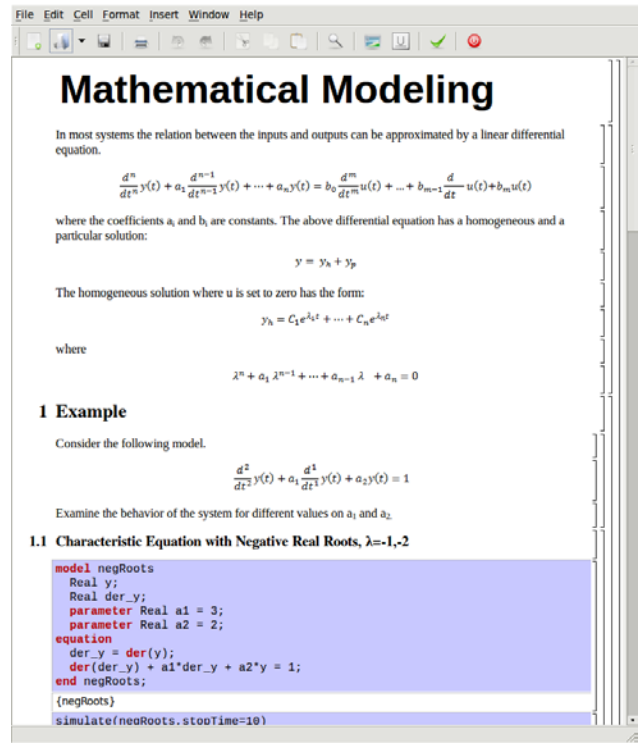


Figure 6. Mathematical modeling.

In the first example the values of a_1 and a_2 are chosen in such way that the characteristic equation has negative real roots and thereby a stable output response, see Figure 7.

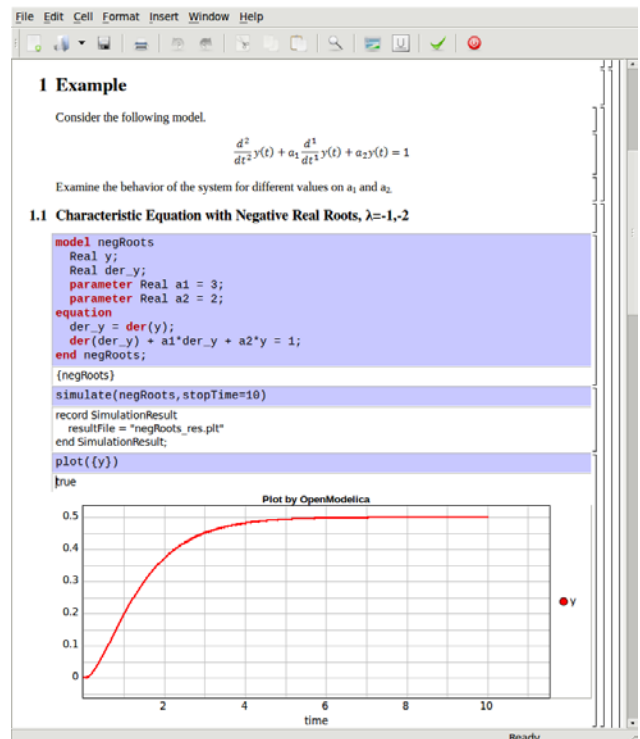


Figure 7. Characteristic eq. with real negative roots.

The importance of the sign of the roots in the characteristic equation is illustrated in Figure 7 and Figure 8, e.g. a stable system with negative real roots and an unstable system with positive imaginary roots resulting in oscillations.

```

model NegRoots
  Real y;
  Real der_y;
  parameter Real a1 = -2;
  parameter Real a2 = 10;
equation
  der_y = der(y);
  der(der_y) + a1*der_y + a2*y = 1;
end NegRoots;
    
```

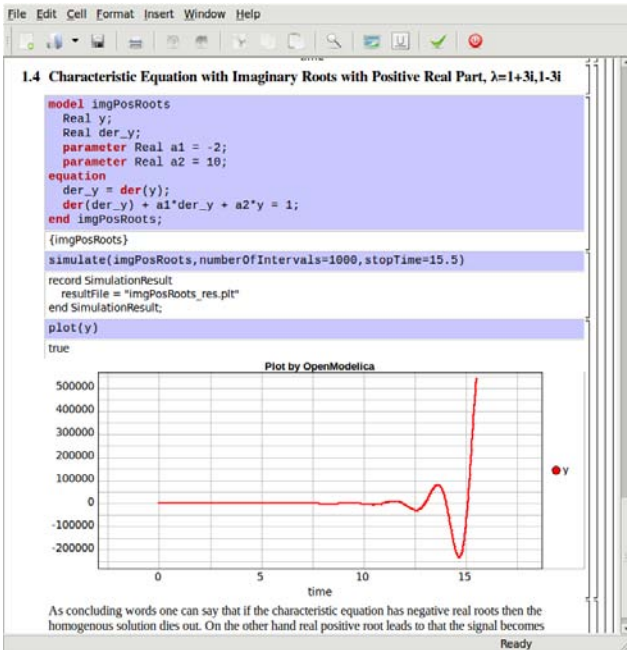


Figure 8. Characteristic eq. with positive imaginary roots.

3.3 Transfer Function

Students also get familiar with how a transfer function, polynomial fraction of the Laplace transform of output over the input, is derived and how it can be used to study the system behavior, see Figure 9 and Figure 10.

The poles of the transfer function are the roots of the denominator which is the same as the roots to the characteristic equation. The zeros are the roots to the numerator of the transfer function. The inverse Laplace transform of $G(s)$ is called the weight function and is the impulse response of the system.

$$Y(s) = G(s)U(s)$$

Lets now look at a simplified first order model of a tank system:

$$G(s) = \frac{1}{s + \frac{1}{T}}$$

```

model Tank
  import Modelica.Blocks.Continuous.*;
  TransferFunction G(b = {1/A}, a = {1,1/T});
  TransferFunction GStep(b = {1/A}, a = {1,1/T});
  parameter Real T = 15 "Time constant";
  parameter Real A = 5;
  Real uStep = if (time > 0 or time<0) then 1 else 0 "step function";
initial equation
  G.y = 1/A;
equation
  G.u = if time > 0 then 0 else 1e6;
  GStep.u = uStep;
end Tank;
    
```

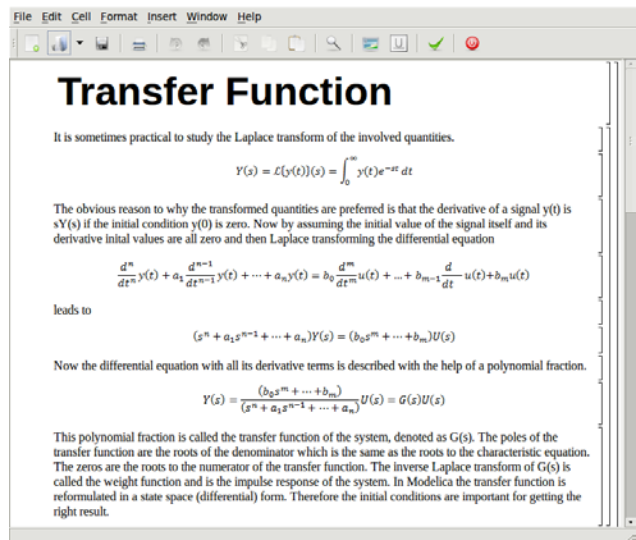


Figure 9. Transfer function derivation.

For analysis of a simple tank model the step and pulse responses of this system are illustrated in Figure 10. In Modelica the transfer function is reformulated in a state space (differential) form. Therefore the initial conditions are important for getting the right result.

The inverse Laplace transform of $G(s)$ is called the weight function and is the impulse response of the system. In Modelica the transfer function is reformulated in a state space (differential) form. Therefore the initial conditions are important for getting the right result.

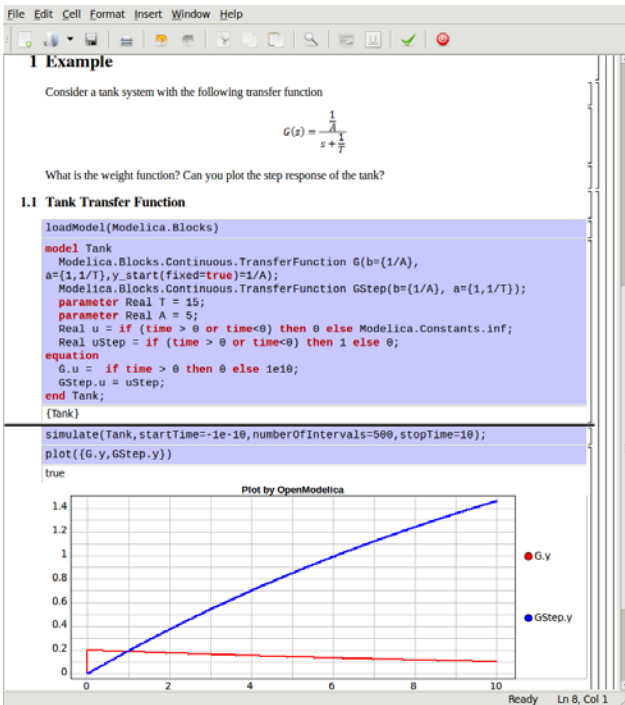


Figure 10. Step and pulse (weight function) response.

3.4 State-space Formulation

The state of a system is the amount of information needed for determining the future output of the system if the future inputs are known.

The state space form for continuous-time dependent systems can be expressed as a system of first order differential equations. We can reformulate the below second order differential equation

$$\ddot{y} + a_1\dot{y} + a_2y = bu$$

by introducing new auxiliary variables

$$\begin{cases} x_1 = y \\ x_2 = \dot{y} \end{cases}$$

the differential equation can be re-written in a state-space form:

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ -a_2 & -a_1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} 0 \\ b \end{pmatrix} u$$

Depending of the modeled system and the type of analysis one would like to perform there could be a desire to shift from the state space formulation to transfer function representation or vice versa.

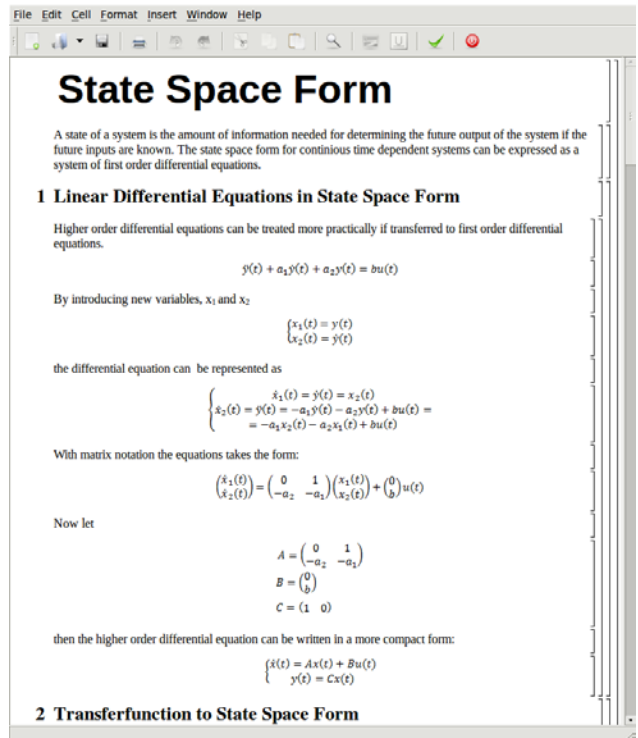


Figure 11. Linear state-space form.

In Figure 12 a second order system is modeled, both with the aid of pure differential equation and also with the transformation to the transfer function representation.

What is important to highlight here is that the two models show different results making the student aware of setting the initial data correctly.

```

model StateSpaceHD
  Modelica.Blocks.Continuous.StateSpace
    stateSpace(A=[-2, 1; -3, 0], B=[-3; 5]
              , C=[1, 0], D=[2]);
  Modelica.Blocks.Sources.Step
    step(height=1.0);
equation
  connect(step.y, stateSpace.u[1]);
end StateSpaceHD;

model DiffEqHD
  Real u = 1;
  Real y;
  Real uprim = der(u);
  Real z = der(y);
equation
  der(z)+2*z+3*y = 2*der(uprim)+uprim+u;
end DiffEqHD;
    
```

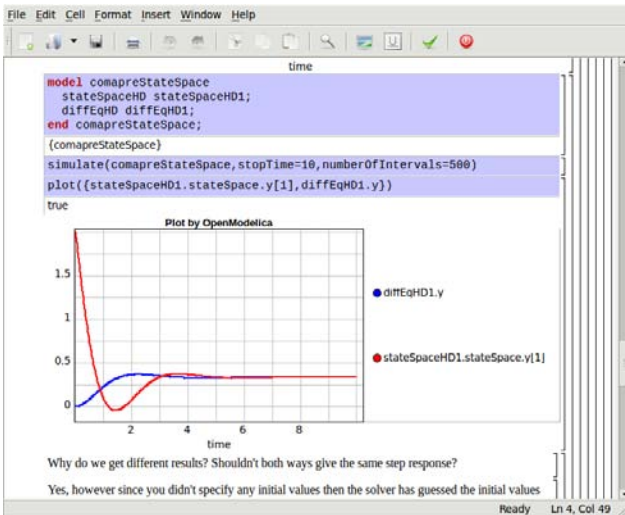


Figure 12. State-space form vs. differential equation modeling.

3.5 Observers and Reconstructed systems

Often we do not have access to the internal states of a system and can only measure the outputs of the system and have to reconstruct the state of the system based on these measurements. This is normally done with an observer, e.g. Kalman filter, see Figure 13 and Figure 14.

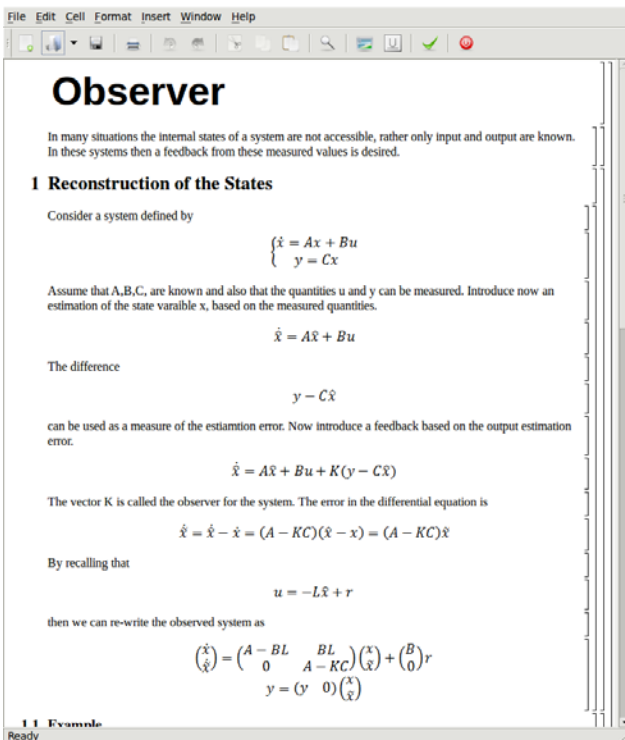


Figure 13. Observer.

Consider the second order model from section 3.4

$$\begin{cases} \dot{x} = Ax + Bu \\ y = Cx \end{cases}$$

Introduce now an estimation of the state variable x :

$$\dot{\hat{x}} = A\hat{x} + Bu$$

The difference

$$y - C\hat{x}$$

can be used as a measure of the error in this estimation.

With the feedback loop

$$u = -L\hat{x} + Br$$

the observed system can be re-written as:

$$\begin{pmatrix} \dot{x} \\ \dot{\hat{x}} \end{pmatrix} = \begin{pmatrix} A - BL & BL \\ 0 & A - KC \end{pmatrix} \begin{pmatrix} x \\ \hat{x} \end{pmatrix} + \begin{pmatrix} B \\ 0 \end{pmatrix} r$$

$$y = \begin{pmatrix} C & 0 \end{pmatrix} \begin{pmatrix} x \\ \hat{x} \end{pmatrix}$$

The vector K is called the observer for the system.

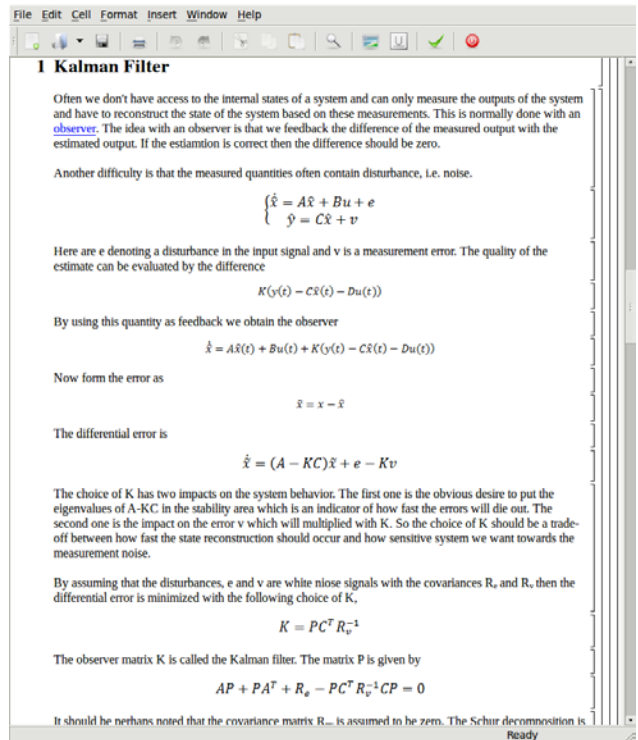


Figure 14. Kalman observer.

In real life systems the observed signals often contain noise. By introducing noise in the observed output signal the modeled system can be made more realistic. The random function is listed below:

```

type Seed = Real[3];
function random
  input Seed si;
  input Real tim;
  output Real x;
  output Seed so;

```

```

algorithm
  so[1] := abs(rem((171*si[1]*exp(
    mod(tim-11,tim+13))),30269));
  so[2] := abs(rem((172*si[2]*exp(
    mod(tim-5,tim+7))),30307));
  so[3] := abs(rem((170*si[3]*exp(
    mod(tim-23,tim+76))),30323));
  if so[1] < 1e-4 then
    so[1] := 1;
  end if;
  if so[2] < 1e-4 then
    so[2] := 1;
  end if;
  if so[3] < 1e-4 then
    so[3] := 1;
  end if;
  x := rem((so[1]/30269.0 +so[2]/30307.0 +
    so[3]/30323.0),1.0);
end random;

```

The time input is needed to ensure that the Modelica compilers shouldn't consider the above function as constant.

Now the state-space model from the Modelica standard library can be re-written containing noise:

```

block StateSpaceWithNoise "Linear state
  space system with noise"
  parameter Real A[:,size(A, 1)] =
    {{0,1},{1,0}} ;
  parameter Real B[size(A, 1),:] =
    {{0},{1}};
  parameter Real C[:,size(A, 1)] =
    {{1,0}};
  parameter Real F[size(A, 1),:] =
    {{1},{0}};
  parameter Real D[size(C, 1),size(B, 2)]
    = zeros(size(C, 1), size(B, 2));
  extends Modelica.Blocks.Interfaces.MIMO(
    final nin = size(B, 2), final nout =
    size(C, 1));
  output Real x[size(A,1)] "State vector";
  Real si(start ={1,2,3});
  Real si2(start={11,27,127});
  Real randomE "input noise";
  Real randomV "measurement noise";
algorithm
  (randomE,si) := random(si,time/10);
  (randomV,si2) := random(si2,time/10);
equation
  der(x) = A * x + B * u + F*{randomE};
  y = C * x + D * u + {randomV};
end StateSpaceWithNoise;

```

```

model StateSpaceNoise
  StateSpaceWithNoise stateSpace;
  Modelica.Blocks.Sources.Exponentials
  ref(outMax=4,riseTime=1,
  riseTimeConst=1,fallTimeConst=0.2,
  offset=0,startTime=-1);
initial equation
  stateSpace.x[1]=1;
equation
  connect(ref.y, stateSpace.u[1]);
end StateSpaceNoise;

```

Lets now look at a simple noisy pendulum model where the output angle is observed with a Kalman observer:

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \end{pmatrix} u$$

$$y = x_1$$

```

model KalmanFeedback
  parameter Real A[:,size(A, 1)] =
    {{0,1},{1,0}} ;
  parameter Real B[size(A, 1),:] =
    {{0},{1}};
  parameter Real C[:,size(A, 1)] =
    {{1,0}};
  parameter Real [2,1] K = [2.4;3.4];
  parameter Real [1,2] L = [2.4,3.4];
  parameter Real[:,:] ABL = A-B*L;
  parameter Real[:,:] BL = B*L;
  parameter Real[:,:] Z =
    zeros(size(ABL,2),size(AKC,1));
  parameter Real[:,:] AKC = A-K*C;
  parameter Real[:,:] Anew = [0,1,0,0 ; -
    1.4, -3.4, 2.4,3.4 ; 0,0,-2.4,1;0,0,
    2.4,0];
  parameter Real[:,:] Bnew = [0;1;0;0];
  parameter Real[:,:] Fnew = [1;0;0;0];
  StateSpaceNoise Kalman(
    StateSpace.A=Anew,
    StateSpace.B=Bnew,
    StateSpace.C=[1,0,0,0],
    StateSpace.F = Fnew);
  StateSpaceNoise noKalman;
end KalmanFeedback;

```

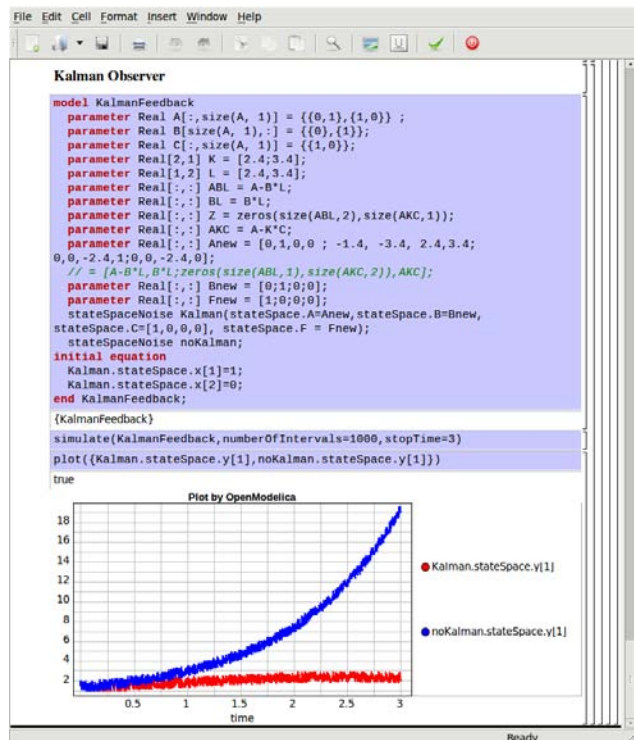


Figure 15. Pendulum angle control with Kalman observer.

3.6 Linear Quadratic Optimization

A good measure of suitable feedbacks, e.g. extra poles, is minimum of the input and output energy levels. Solving the minimum energy level functional leads to the algebraic Riccati equation, shown in Figure 16 and Figure 17.

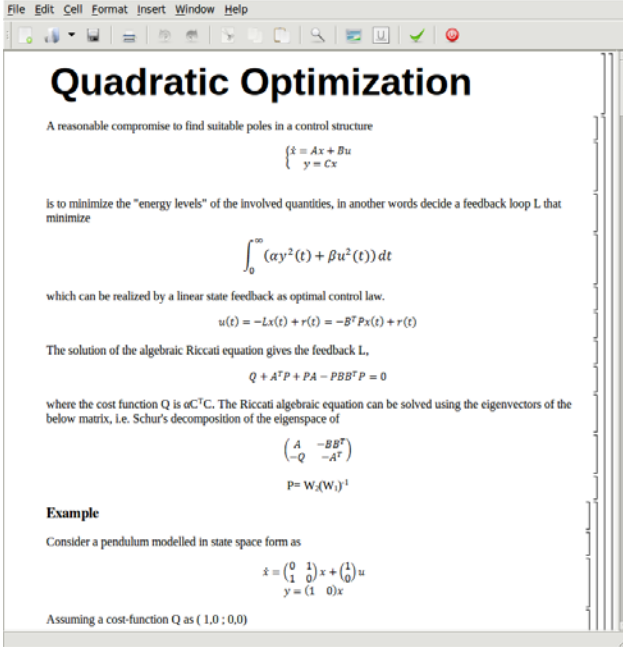


Figure 16. Quadratic optimization.

The algebraic Riccati equation is:

$$AP + PA^T + R_e - PC^T R_v^{-1} CP = 0$$

```

model RiccatiEq
  parameter Real A[2,2]=[0,1; 1,0];
  parameter Real B[2,1]=[0; 1];
  parameter Real C[1,2]=[1,0];
  Real P[2,2](start = Pinit);
  parameter Real Pinit[2,2] =
    [1,1.5;1.5,1];
  parameter Real Q[2,2] = [1, 0; 0, 0];
  Real L[1,2];
  Real L1 = L[1,1];
  Real L2 = L[1,2];
equation
  Q + P*A + transpose(A)*P -
    P*B*transpose(B)*P = [0,0;0,0];
  L = transpose(B)*P;
end RiccatiEq;
    
```

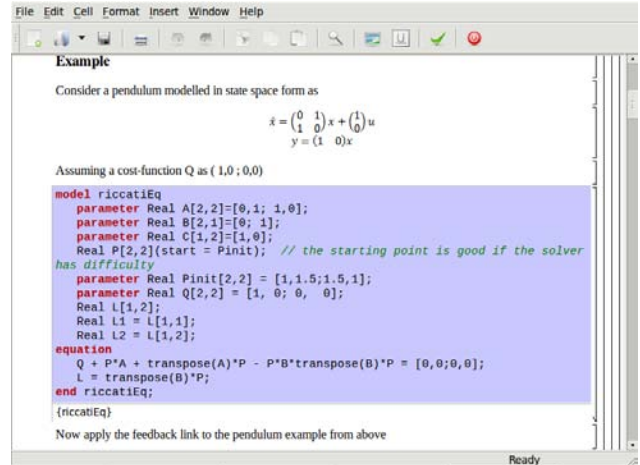


Figure 17. Solving Riccati equation with OpenModelica.

3.7 Linearization

Many nonlinear problems can be handled more easily by linearization around an equilibrium point. Lapunov showed that if the linear approximation is stable then the nonlinear problem is also stable, at least around the equilibrium region. Thus we can investigate the behavior of the nonlinear system by analyzing the linearized approximation.

In the OpenModelica Compiler (OMC) a flag is introduced for linearization:

```
setCommandLineOptions({" +d=linearization" })
```

Assume that we have a non-linear two tank model shown below:

```

model TwoFlatTankModel
  Real h1(start = 2);
  Real h2(start = 1);
  Real F1;
  parameter Real A1 = 2,A2 = 0.5;
  parameter Real R1 = 2,R2 = 1;
  input Real F;
  output Real F2;
equation
  der(h1) = (F/A1) - (F1/A1);
  der(h2) = (F1/A2) - (F2/A2);
  F1 = R1 * sqrt(h1-h2);
  F2 = R2 * sqrt(h2);
end TwoFlatTankModel;
    
```

The output C-files are now generated with the linearization flag. By running the executable with the time argument the linearized model is generated which can be simulated:

```

buildModel(TwoFlatTankModel) //OMC
system("TwoFlatTankModel.exe -l 0.0 -v
  >log.out")
readFile("log.out")
    
```

The file `log.out` contains now the linearized model:

```

model Linear_TwoFlatTankModel
  parameter Integer n = 2; // states
  parameter Integer k = 1;
  parameter Integer l = 1;
  parameter Real x0[2] = {2,1};
  parameter Real u0[1] = {0};
  parameter Real A[2,2] = [-0.5,0.5;2,-3];
  parameter Real B[2,1] = [0.5;0];
  parameter Real C[1,2] = [0,0.5];
  parameter Real D[1,1] = [0];
  Real x[2](start = x0);
  input Real u[1](start = u0);
  output Real y[1];
  Real x_Ph1 = x[1];
  Real x_Ph2 = x[2];
  Real u_PF = u[1];
  Real y_PF2 = y[1];
equation
  der(x) = A * x + B * u;
  y = C * x + D * u;
end Linear_TwoFlatTankModel;
    
```

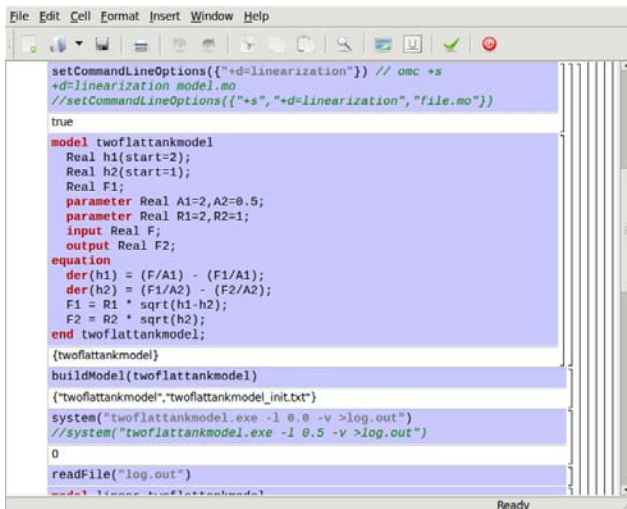


Figure 18. Linearization.

4 Other OMNotebook Applications

OMNotebook is also used for teaching modeling with Modelica (DrModelica), and programming in Scheme (DrScheme).

4.1 DrModelica

The existence of numerical algorithms and solvers are important aspects of equation-based environments such as Modelica tools.

OMNotebook is currently being used for course material (DrModelica) in teaching the Modelica language and equation-based object-oriented modeling and simulation, (see Figure 19).

It can easily be adapted to electronic books teaching other programming languages, such as Scheme (Sec-

tion 4.2). OMNotebook can also easily be used in other areas such as physics, biology chemistry, biomechanics etc., where phenomena can be illustrated by dynamic simulation within the book.

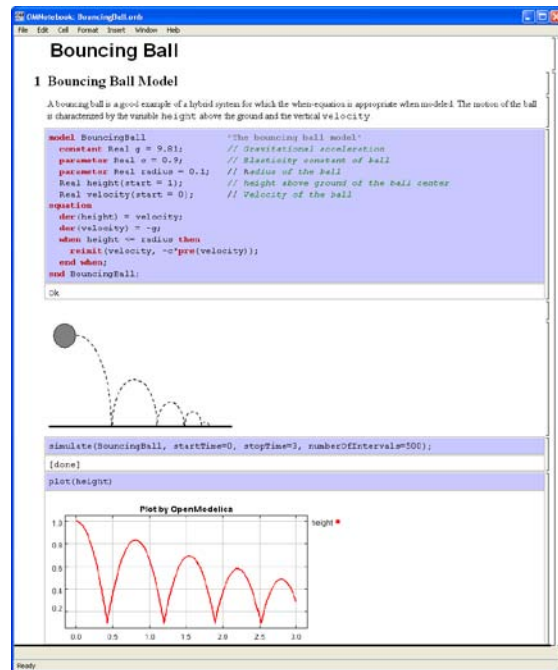


Figure 19. Bouncing ball example with movement animation in OMNotebook.

4.2 OMScheme

With OMScheme the OMNotebook paradigm is generalized towards other programming languages than Modelica, e.g. the Scheme programming language, [6]. An implementation of the factorial function using OMScheme is shown in Figure 20.

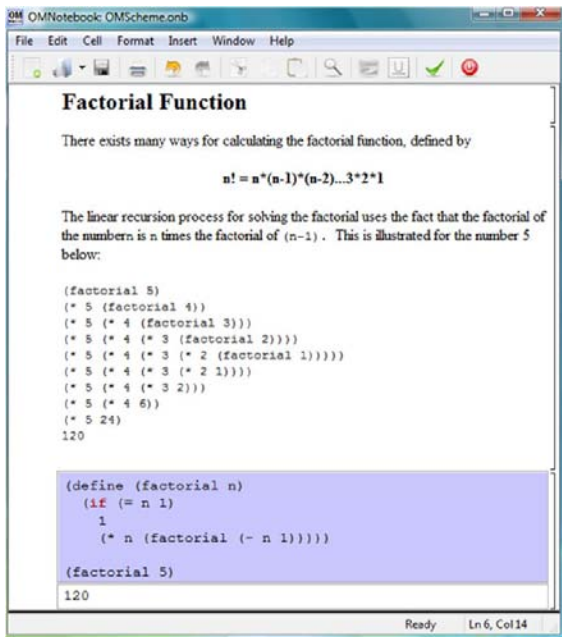


Figure 20. Factorial function in OMScheme.

5 Future Work

The inherent features of the Modelica language makes the next mile-stone choice quite natural, namely the adaption of the presented concept into other engineering courses as well, e.g. a future course material called DrMechanics for teaching the basics of mechanical systems.

A future generation of OMNotebook is planned to be extended to become available through a web applet which would make the material available without needing installation of any software.

One thing that is intentionally left out in this paper is frequency domain analysis, e.g. bode diagram. This is partly due to the inherent properties of the Modelica language, which is quite time domain dominant in its modeling style. A work-around was shown in this paper when studying the weight function and step response in time domain.

6 Conclusions

The OMNotebook is one of the first open source efforts offering interactive electronic books for teaching and learning modeling and programming.

In this paper we present its use in an active electronic book called DrControl for teaching control theory and applications.

The idea of active electronic books in OpenModelica has so far been employed in the two E-courses DrModelica and DrControl used successfully in graduate and workshop courses.

7 Acknowledgements

This work has been supported by EU project Lila and Vinnova in the ITEA2 OPENPROD project. The Open Source Modelica Consortium supports the OpenModelica work.

References

- [1] Peter Fritzson. *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*, 940 pages, Wiley-IEEE Press, 2004.
- [2] Modelica Association. *The Modelica Language Specification Version 3.2*, May 2010. <http://www.modelica.org>.
- [3] Anders Sandholm, Peter Fritzson, Varun Arora, Scott Delp, Göran Petersson, and Jessica Rose. The Gait E-Book - Development of Effective Participatory Learning using Simulation and Active Electronic Books. In *Proceedings of the 11th Mediterranean Conference on Medical and Biological Engineering and Computing (Medicon'2007)*, Ljubljana, Slovenia, June 26 - 30, 2007.
- [4] Eva-Lena Lengquist Sandelin, Susanna Monemar, Peter Fritzson, Peter Bunus. DrModelica – A Web-based Teaching Environment for Modelica, In *Proc. of the 44th Scandinavian Conference on Simulation and Modeling (SIMS2003)*, Västerås, Sweden, 2003
- [5] Anders Fernström, Ingemar Axelsson, Peter Fritzson, Anders Sandholm, Adrian Pop. OMNotebook – Interactive WYSIWYG Book Software for Teaching Programming. In *Proc. of the Workshop on Developing Computer Science Education – How Can It Be Done?*. Linköping University, Dept. Computer & Inf. Science, Linköping, Sweden, March 10, 2006
- [6] Mohsen Torabzadeh-Tari, Peter Fritzson, Adrian Pop, Martin Sjölund, *Generalization of an Active Electronic Notebook for Teaching Multiple Programming Languages* IEEE EDUCON Education Engineering 2010 – The Future of Global Learning Engineering Education, Madrid, Spain, 2010
- [7] <http://www.openmodelica.org> [accessed 2011-02-03]

Modelica Simulator Compatibility - Today and in Future

Jörg Frochte

Hochschule Bochum

Department of Electrical Engineering and Computer Science

Höseler Platz 2, 42579 Heiligenhaus, Germany

email: joerg.frochte@hs-bochum.de

Abstract

In this paper we would like to give a small snapshot in time on Modelica tool compatibility today, and discuss strategies for its improvement in order to keep it on a high level. Especially we would like to consider approaches for semi-automatic test and verification frameworks as well as to develop different levels and definitions on Modelica tool compatibility.

Keywords: Modelica language; simulation and design tools; quality and compatibility management

1 Introduction

The Modelica language grows as well in complexity as in scope and becomes a mighty tool to describe models from very different domains. Beyond this the number of tools using the Modelica language has increased to the benefit of the users. A various set of compatible simulators decreases the dependency on a single tool provider, allows exchange between different modelers using different tools, and – because every development of a model and its maintenance is not for free – it offers a high degree of investment security.

But there are dangers to be avoided as well. A diverging interpretation of a standard and a heterogeneous set of vendors may lead to unpleasant scenarios for users and library providers.

Compatibility is not an easy task for a complex declarative language which is the base of model-based software generation. Beyond the language and its interpretation the results given by the generated code may differ based on various parameters like the chosen integrator and its boundaries for relative and absolute errors. So some variations in the results are allowed and expected, some are not.

Drawing a comparison with C++-Compilers it is expected that the quality of the generated executable differs, e.g. concerning performance. It might be ac-

cepted that not every compiler can handle any part of the new C++ standard, but it falls beyond the pale, if source code is successfully compiled and executed, and finally provides totally different results based on the used compiler.

In order to receive an impression how compatible Modelica tools are among themselves and as well among the standard we took a snapshot of four tools, mostly demo and testing versions, and gave them quite tiny models that can be translated with limited versions.

For us the fact which tool can handle what kinds of models is of minor interest, because this will properly change with every new version. More important for us is the answer to the question if there is already a diverging interpretation of the standard or if this issue has been of no interest so far. In this case we could assume that there will probably be a growing need for measures guaranteeing quality as well as compatibility, which have to be presented later on.

2 Snapshot on Modelica tool compatibility

Let us start with a scenario in which two developers are using different tools to model and simulate. Now developer A and B exchange models, given in Modelica source code, and naturally they both expect no problems, because they meet the Modelica standard. Because of the grown complexity and increasing number of interpretations of the standard this may run ill and so it may turn out that an easy exchange is not possible. We distinguish between three different kinds of causes presented in the following sections.

2.1 Lacking support of language elements

The lacking support of language elements is easiest to handle for the users. In this case the translation of the Modelica model will fail and provide a hint which language element is not supported. In a

lot of Modelica tools a common non-provided language element is the *else-when*-element in the equation section of a model. This type of compatibility problem is annoying but it causes no danger.

A good example is the following model:

```

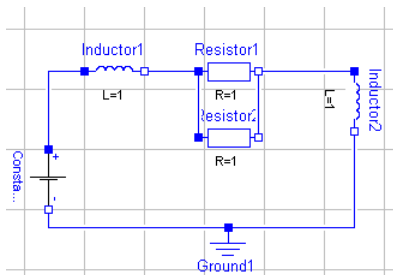
model A
  Real x;
  Real a(start=2);
equation
  when {time > 0.5,x > 0.7} then
    if time > 0.55 then
      a = 3;
    else
      a = 4;
    end if;
  elseif time > 0.6 then
    a = 5;
  end when;
  der(x) = 1;
end A;
    
```

Model 1: *else-when* in equation section

It turned out that in our snapshot of four Modelica tools only one could translate and simulate the model above. Two of them were not able to translate it and one generated code but directly aborted using the compiled program.

2.2 The generated code

The next cause has its roots in the different skills of the tools to optimize code and handle tricky situations concerning e.g. index reduction. Most tools use a more or less straightforward implementation of the Pantelides Algorithm [4], extended with dummy derivatives (s. e.g. [5] or [1], chapter 7). In some situations this is not always enough and may lead to non-executable results. For example, this combination of inductors and resistors cannot be simulated in every Modelica tool:



Model 2: *advanced index reduction*

These kinds of compatibility problems are in a way in between, if they are real compatibility issues at all. The quality of code generation and GUI is the reason

for a user choosing one tool or another. So on one side diverging results in quality should be expected and respected, but in any case it would be interesting to measure it. On the one hand the tool provider is presumably keen on increasing the quality of his product. And on the other hand, if the results were published, it may even help the customer to choose a Modelica tool. So we think these are the kinds of variations in the results that are allowed, expected, and do not touch the goal of standardized and compatible language.

However, if the code is executable and runs without a warning, which was not the case for any tested tool, this scenario comprises some risks. So like it seems to be now, a termination of the simulation as soon as possible should always be the default handling of such cases. Thus we assume that in most cases the simulation will be directly aborted if such a problem arises.

2.3 Different interpretation of language element

In contradistinction to lacking support of language elements or variations in the code quality a different interpretation of language elements of the various Modelica tools may lead to more serious problems. Let us have a look at the following example:

```

class A
  class C
    Real t(start=-2);
    Real x;
  equation
    der(x) = t;
  algorithm
    when time > -0.1 then
      t := time + 0.1;
    end when;
  end C;
  C c;
end A;
    
```

Model 3: *when and HDAE initialization (I)*

All of the provided examples are very small and can be evaluated in most test or demo versions of Modelica tools. From a theoretical point of view and our interpretation of the Modelica standard [5] the initial value of t should be -2 . In our test it turned out that just one of the tested Modelica tools computed the result -2 and most of them started with $t = 0.1$. Independent of the correct value, the interesting effect for us is that the whole simulation may run differently now. In the next model, the problems might

have their cause in interpretation or implementation of initial conditions and equation reduction.

```

model A
Real x(start=7);
Real y,z;
flow Real a[2,3];
equation
z = -a[2,3];
z=x;
y=z;
a[1,1] = 0;
a[1,2] = 0;
a[1,3] = 0;
a[2,1] = 0;
a[2,2] = 0;
der(a[2,3]) = 1;
end A;
    
```

Model 4: Transfer of start values

In our interpretation the resulting equation $x=y$ should propagate the initial value, so that the simulation starts with $x=y=z=7$ and $a[2,3]=-7$.

Our point of view is that a consistent initial value should always be tried to be propagated directly, independent of an additional attribute like *fixed=true*.

But in some tools start values are not propagated and so the simulation starts with different initial values. The effect is of course a simulation with totally different results. A possible explanation might be that some tools call the initial function with $x=y=z=7$ and $a[2,3]=-7$ and try to find a consistent set of variables based on these initial values. Whereas the other tools start with this procedure with $x=7$ and assume that this information will be propagated during the DAE initial problem.

Finally let us look at the following very small model 5:

```

model A
Real x(start=2);
Real y(start=3);
equation
der(x) = 1;
when x > 1 then
y = pre(y) + 1;
end when;
end A;
    
```

Model 5: when and HDAE initialization (II)

In the Modelica Standard 3.2 ([5]), section 8.3.5 the language element is defined as follows: “The statements within a *when*-equation are activated

when the scalar expression or any of the elements of the vector expression becomes true.” So it could be translated in an *if*-condition like this:

```

if (boolean) and not pre(boolean)
    
```

In combination with the techniques for an initialization after an event indicated on page 226 of [5] - the init situation is slightly the same - we come to the conclusion that the condition should only be activated, if $x>1$ has been logical false before and is now logical true. This situation is quite tricky for a lot of Modelica tools. We think that x is equal to 2 immediately, and the state x has a positive derivative during the whole time. Therefore, the *when*-cause should never be activated and so y is equal to 3 all the time. Just one Modelica simulator handled the situation like this, all the others computed $y=4$.

So we have got two situations: model 3, in which the time value has to be interpreted, and model 5, which includes an initial value for x . In both situations the *when*-language element is misinterpreted during the initial phase. In case of model 5 one might argue about the start value of x , but obviously the time value as in model 3 is never negative.

We tested some more models, and as well as in the presented ones we found a few similar results. So we can conclude that there are diverging interpretations of the standard and that it makes sense to think about strategies in order to avoid this.

3 Suggestion of semi-automatic test and verification frameworks

The first step in our strategy is briefly presented in the following figure:

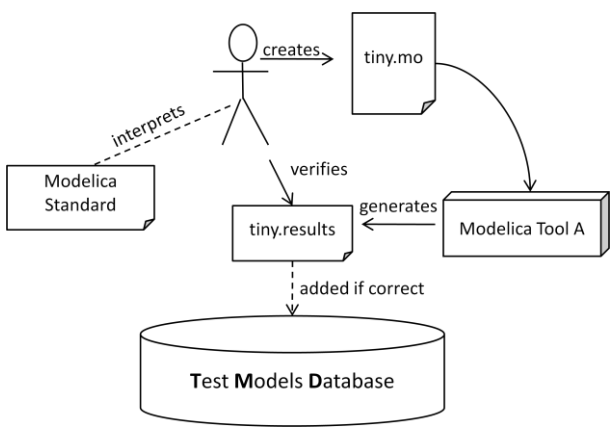


Figure 1: Test Database and models with proved results

The most important aspect could be to create a database with tiny Modelica models which contain a single element of the Modelica standard one would like to have tested. The models themselves should be tricky for the translation process, but so small that the results can be exactly verified by a human based on the common interpretation of the Modelica standard. This is very important because certainly it is not proved that Modelica tool A in the figure is free from errors.

These tiny models are in a way academic and their benefit is that the correctness of the results can be proved. When a correct pair of model and result has been created, both are added to the database.

This is the most important aspect for a check for compatibility. But it is hard to collect a lot of these models and it might not be possible to get all complex side effects. So beyond this, one should add “applied models” to the database:

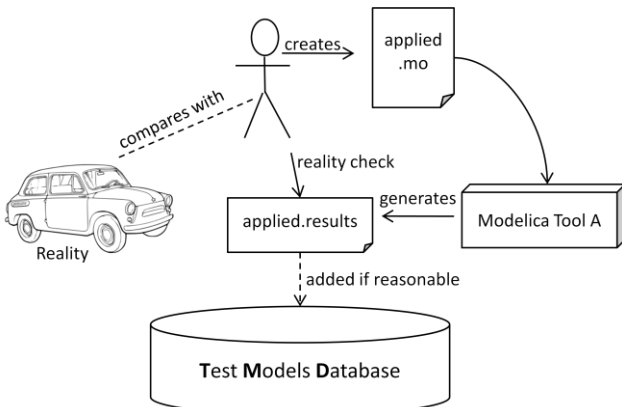


Figure 2: Test Database and applied models with reasonable results

In general, no human can prove the correctness of a complex model like this. So in this case it is just a check if the results are reasonable or not. If they are reasonable the reference results as well as the model are added to the database. In the next step it makes sense to distinguish between these different kinds of models:

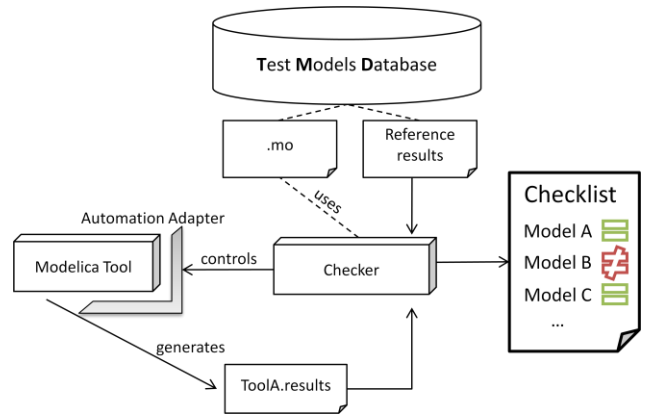


Figure 3: Semi-automatic test scenario

With this database it would now be possible to test the Modelica tools on Standard compatibility at least semi-automatically. A checker program can pick up the Modelica model with its associated reference results, simulate the model with the Modelica tool under test and compare the results. The comparison of the results needs a little bit a fuzzy approach, because it is unlikely that the generated code will produce totally the same results as in the reference solution. There are a lot of interesting approaches, see e.g. [8], to compare models and their results in a (semi)-automatic way. For simplicity let us assume that all variables of the model are stores in a vector \mathbf{x} and the corresponding reference solutions in a vector \mathbf{r} , then one might check for $|\mathbf{x}_i(t) - \mathbf{r}_i(t)| < tol$. Another important benchmark value should be the time of the event as well as the initial values after the event.

If every compiled Modelica code uses a different integrator for the HDAE, this might lead to serious problems for the compression, so just for this benchmark purpose one should fix a freely available Open Source integrator like IDA (s. e.g. [3]). With a fixed integrator such a test should be possible.

But why do we call this a semi-automatic test for it looks like a full-automatic test until now? Responsible for this are the failures which may occur when simulating or translating the “applied models”. In this case it is not possible to exclude the possibility that the reference results are wrong. So in some cases it might be necessary to reevaluate the reference results for such a model.

4 Levels of compatibility and quality control

With such an infrastructure a validation service could be set up. Nevertheless, this would just produce a list of successfully and not successfully handled models, but no solution to the fundamental problem. So at that point one would be able to measure the language respectably the tool compatibility and achieve information about what to improve. But this is just one aspect of two. The two challenges for the compatibility issue on the long run are firstly the growing complexity and secondly measurement and control. But up to now we majorly dealt with the measurement aspect. How could growth of complexity be reduced without acting as a brake upon new innovations in the Modelica language?

A possible approach might be introducing different levels of complexity in the Modelica language. The highest level could be today's Modelica language with all its language elements as well as upcoming innovative and progressive features. The lower levels should be becoming more and more conservative concerning changes and less complex:

1. (Full) Modelica
2. Simple Modelica
3. Flatmodelica

One might think that Flatmodelica already exists for very often the term "flat model" is mentioned, and therefore one has an association, because e.g. some tools allow exporting "Flatmodelica" or a "flat model". But this is not true, for there is no official standard defining Flatmodelica or at least a "flat model". What comes close to a definition is written in [2], chapter 18, but anyway it is not a formal language description. The name suggests that there are no more dependencies concerning libraries. Beyond this, it implies that no more inheritance has to be carried out, but this is hard to be done, because nearly everything in Modelica is a class. So e.g. the question occurs whether records are part of Flatmodelica or not.

Maybe we should motivate why it could make sense to introduce different levels of complexity and development speed in the Modelica standard.

The major reason could be that no tool has been able to implement the full Modelica language up to now. If the language keeps on growing, as during the course of recent years, it is hardly probable that this status will change.

To illustrate this, let us have a look at the following model:

```
class A
  Real x[2, 3];
  Integer i=7;
  Integer j=8;
  Real y;
equation
  for i in {1,2}, j loop
    x[i, j] = i*A.j;
  end for;
  for i in {3,4}, j loop
    x[i, j] = A.i*j;
  end for;
  y=time*x[2,2];
end A;
```

Model 6: Automatic detection of array bounds

This model exclusively uses valid Modelica language, but in our tests none of the tools has been able to generate code and simulate it successfully. The used language elements are not new in Modelica 3.2, so we can exclude the effect that the tools were unable to implement them in time. An explanation might be the high demands of such dynamic language elements for the data structures of a Modelica tool. This model is not an isolated incident; let us for example regard this model:

```
model A
  record R
    Real x[1,1];
    Boolean b;
  end R;
  Real w;
  R r1(x={{1}}, b=false);
  R r2(x={{2}}, b=true);
  R r3[2];
equation
  r3[1] = r1;
  r3[2] = if time > 0.5 then
    (if time > 0.6 then r1 else r2)
  else r2;
  der(w) = r3[2].x[1,1];
end A;
```

Model 7: Usage of records

Records as well as the automatic detection of array bounds and especially their dynamical handling cause just one problem in many tools. The effects probably differ because of the different data structures used to translate the Modelica models.

So we can conclude that full Modelica, as fast developing language standard, is not predestinated as cross-tool exchange language. Beyond these features and aspects, there are a lot of chapters in the Modelica standard that could likely be excluded for a Simple Modelica approach. Examples might be [5], section 10.5.1 “Indexing with Boolean or Enumeration Values”, chapter 14 “Overloaded Operators”, some of the redeclaration features described in chapter 7.3 or the expandable connectors from chapter 9.3.1.

Such features proposed to be excluded from Simple Modelica in comparison to full Modelica are mostly the very dynamic ones and therefore hard to be validated using the proposed semi-automatic test and verification framework, especially but not only concerning more complex models. In fact the suggested test and validation infrastructure will be more efficient on the lower levels and always less efficient on the higher ones. One reason would be the conservative progressing approach and another one limited amount of language elements and features.

Because the mentioned features are apparently difficult to be implemented for Modelica tool vendors, it would probably be possible to achieve a better tool compatibility on the lower levels compared to the higher ones. Beyond this, the introduction of less featured levels of Modelica might even lead to a provision of a base for the exchange of Simple or Flatmodelica models to and from the proprietary Simscape language, s. e.g. [8], invented by TheMathWorks. To support such a scenario for Simple Modelica the redeclaration techniques described in [5], section 7.3, might need further restrictions or simplifications. Anyway it is of course unlikely that this will work without a conversion procedure, but a conversion from Simple Modelica to Simscape might be possible, while the more complex and mighty Full Modelica is a formidable challenge for an automatic conversion from and majorly to Simscape.

So Simple Modelica and Flatmodelica as subsets of full Modelica could provide grand strides concerning cross-tool exchange, tool compatibility and finally make formal tests discussed in section 3 much more efficient.

If we had these two subsets of Modelica we could judge tools by their capacity to import, export and translate models on three different levels. To achieve a simple measurement for users one may introduce a bronze, silver, gold and platinum tag on the different levels. The platinum tag will just be given, if a tool

can handle the full test without failures, the gold tag with a given percentage and so on. The suggested infrastructure could be set up by a central organization like the Modelica Association.

As discussed above it is very unlikely that a tool would reach the platinum level for the latest few full Modelica language versions. Most tools could achieve gold and silver, but obviously this would not be the perfect exchange level because every tool might miss different language aspects. On the more conservative lower language levels like Simple Modelica or Flatmodelica a lot of tools could reach platinum level and therefore provide a good base for a cross-tool exchange with a consistent language interpretation.

For a kind of “flat model”, like e.g. in Dymola, can be generated from full Modelica without any loss of functionality it should be possible to do the same with a formal defined Flatmodelica and a Simple Modelica. So there won’t be any loss of functionality, just a loss of structure and convenience. This is the reason why Simple Modelica as intermediate stage between full Modelica and Flatmodelica makes sense. Flatmodelica as kind of textual description of an HDAE is always possible, but it is hard to maintain a model described in Flatmodelica while it is hard to achieve a high compatibility level for full Modelica. So Simple Modelica together with a semi-automatic test and verification framework could lead to a high degree of investment security and independence for users and library providers.

5 Conclusions

So finally we conclude that the desirable growth of Modelica tool vendors and language capacity leads to a lot of benefits but also to the issue of compatibility which today and in future will become more and more important.

We have shown that recently there is a need to introduce quality control mechanisms. Beyond this, we tried to give brief suggestions how it might be possible to deal with the task of compatibility of Modelica tools among themselves and as well among the Modelica standard.

References

- [1] F. E. Cellier und E. Kofman. **Continuous System Simulation**, Springer (2006)
- [2] P. Fritzon. **Principles of Object-Oriented Modeling and Simulation with Modelica 2.1**, John Wiley & Sons (2004)
- [3] A. C. Hindmarsh, P. N. Brown, K. E. Grant, S. L. Lee, R. Serban, D. E. Shumaker and C. S. Woodward: **SUNDIALS: Suite of Nonlinear and Differential/Algebraic Equation Solvers**. In ACM Transactions on Mathematical Software, 31(3), pp. 363-396, 2005.
- [4] Mattsson and Söderlind. **Index Reduction in Differential-Algebraic Equations using Dummy Derivatives**, SIAM J. SCI. COMPUT. Vol.14. No.3, pp. 677-692 (1993)
- [5] *Modelica Association: Modelica Language Specification Version 3.2*; (March 2010)
- [6] C. C. Pantelides. **The Consistent Initialization of Differential-Algebraic Systems**. In SIAM J.Sci.Stat.Comput. Vol.9, No. 2, (1988)
- [7] B. Stein. **Model Compilation and Diagnosability of Technical Systems**. In 3rd Int. Conference on Artificial Intelligence and Applications, pp. 191-197 (2003)
- [8] *TheMathWorks: Simscape 3 Language Guide* (March 2010)

A Modelica model of thermal and mechanical phenomena in bare overhead conductors

Oscar Duarte

Universidad Nacional de Colombia, Department of Electrical and Electronics Engineering
ogduartev@unal.edu.co

Carrera 30, Calle 45, Ed. 453, Of. 202. Bogotá, Colombia

Abstract

A Modelica model of the thermal and mechanical phenomena of bare overhead conductors is presented. The geometry of the catenary is also modeled as an important part of the mechanical phenomena. The model has been compiled and tested using OpenModelica. Some application examples are also presented to illustrate some analysis that can be done with the model.

Keywords: overhead conductors, IEEE 738, catenary, sag

1 Introduction

Bare overhead conductors are essential in transmission and distribution of large amounts of electrical energy. They are supported by transmission towers and exposed to varying weather conditions. The mechanical behavior of these suspended cables is affected by thermal processes. Heat transfers change conductor temperature, causing length and tension modifications. As a result, the geometry of the catenary described by the cable also changes. It is very important to study this geometric modifications, mainly to avoid electrical failures caused by the violation of security distances.

In this paper we show a Modelica model of the thermal and mechanical phenomena of bare overhead conductors; the geometry of the catenary is also modeled as an important part of the mechanical phenomena. The model has been compiled and tested using OpenModelica ([2], [3]). Some application examples, and the source code are available at the Virtual Academic Services of the National University of Colombia ¹.

By using Modelica two main advantages have arisen:

- The solution of the state change equation is very simple. This is an equation that must be solved by numerical methods. The Modelica model of the equation is done in a natural way, and the algorithms available in OpenModelica are capable to solve it.
- The sagability analysis is immediate. The most common models from overhead conductors use the electrical current in the conductor to compute the conductor temperature for extreme operation conditions and then the mechanical and geometric variables. Sagability analysis is the inverse process: from the mechanical, geometric or thermal limit conditions we must find an electrical current that will cause them. Due to the object-oriented modeling of Modelica, the same model can be used in both directions.

This paper is organized as follows: in section 2 the physical model is presented in two steps, the thermal model first (section 2.1) and then the mechanical model (section 2.2); in section 3 the Modelica implementation is explained also in two steps (sections 3.1 and 3.2); in section 4 we show some application examples to illustrate the analysis capabilities of the implementation. Conclusions and future work are summarized in section 5.

2 Physical model

2.1 Thermal model

The thermal model calculates the conductor temperature for a certain electrical current and weather conditions. The IEEE Standard 738 ([1])

¹<http://www.lab.virtual.unal.edu.co>

defines two different models: one for steady state calculations and another one for transient calculations.

The non-steady heat balance is summarized by equation 1

$$\frac{dT_c}{dt} = \frac{1}{mC_p} [RI^2 + q_s - q_c - q_r] \quad (1)$$

Where:

- T is the conductor temperature.
- mC_p is the heat capacity of conductor.
- R is the electrical linear resistance of the conductor, which is a function of its temperature.
- I is the current passing through the conductor. The term RI^2 is the heat gain by Joule effect.
- q_s is the heat gain by solar radiation.
- q_c is the heat loss by convection.
- q_r is the heat loss by radiation.

The standard also establishes detailed models for every term in equation 1. The main features of these models are:

R model: electrical linear resistance of conductor is calculated as a function of temperature by a linear interpolation (or extrapolation) of the values at two different temperatures (usually 25°C and 75°C). These values are available in manufacturers data sheets.

mC_p model: total heat capacity is calculated taking into account the percentage of materials in the conductor (usually steel and aluminium).

q_s model: the actual heat gain by solar radiation depends on the geographical latitude and altitude in which the conductor is placed, the day of the year, the time of the day, the absorptivity and size of conductor; two atmosphere conditions are considered: clear and industrial.

q_c model: natural and forced convection must be calculated. The greatest of the two is used in equation 1. The wind velocity and direction is considered; air density and viscosity depends on air temperature. Temperature, size and azimuth of conductor are also involved in the model.

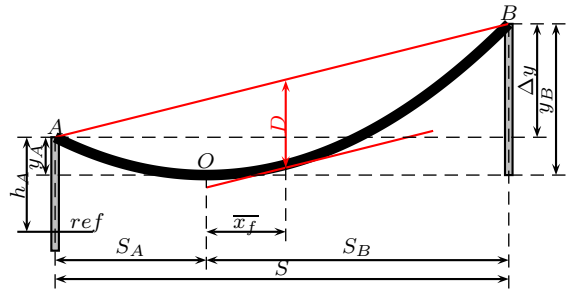


Figure 1: Geometry of the catenary

q_r model: radiated heat is calculated using temperature, emissivity and size of conductor as well as air temperature.

Notice that T , which seems to be explicitly calculated in 1 is really involved in all the terms. A more accurate equation should be:

$$\frac{dT}{dt} = \frac{1}{mC_p} [R(T)I^2 + q_s(T) - q_c(T) - q_r(T)] \quad (2)$$

2.2 Mechanical model

Mechanical phenomena in overhead conductors are well known (see for example chapter 14 of [4]). Figure 1 shows an overhead conductor that describes a catenary. It is supported in A y B , which are at height y_A and y_B from the level of the lowest point (O), and with a difference of level Δy . h_A is the height of A from a reference level. The span (horizontal separation between supports) is S . The longitudinal tension is T_{en} , where as the horizontal tension is H . Conductor length is L , its linear weight is W and its temperature is T . The lowest point O is at a horizontal distance S_A and S_B from supports A and B . The length of conductor from the supports A and B to the lowest point O are L_A and L_B .

Sag D is the maximum vertical distance between the imaginary line that connects A and B and the catenary. Sag occurs at the point in which the catenary tangent is the same as the slope of that imaginary line. The horizontal distance between this point and the lowest point O is \bar{x}_f .

2.2.1 Geometric considerations

The height of the catenary $y(\bar{x})$ from the level of the lowest point O is:

$$y(\bar{x}) = \frac{H}{W} \cosh\left(\frac{W\bar{x}}{H}\right) - \frac{H}{W}; \quad (3)$$

Where \bar{x} is the horizontal distance to O . Let x be the horizontal distance to A . Then we have:

$$\begin{aligned} \bar{x} &= S_A - x \\ y(x) &= \frac{H}{W} \cosh\left(\frac{W(S_A - x)}{H}\right) - \frac{H}{W} \end{aligned} \quad (4)$$

The height of the conductor in x from the reference level is

$$h(x) = h_A - Y_A + y(x) \quad (5)$$

$$h(x) = h_A - \frac{H}{W} \cosh\left(\frac{WS_A}{H}\right) + \frac{H}{W} \cosh\left(\frac{W(S_A - x)}{H}\right) \quad (6)$$

The horizontal distance from the lowest point O to the lowest support is S_{PB}

$$S_{PB} = \frac{S_A}{2} - \frac{H}{W} \sinh^{-1} \left\{ \frac{\Delta y / 2}{\frac{H}{W} \sinh\left(\frac{WS_A}{2H}\right)} \right\} \quad (7)$$

Notice that S_{PB} is equal to S_A or S_B :

$$S_A = \begin{cases} S_{PB} & \text{if } \Delta y \geq 0 \\ A - S_{PB} & \text{if } \Delta y < 0 \end{cases} \quad (8)$$

2.2.2 Computing the sag

In order to find the sag, first we find the slope m of the imaginary line that connects A and B :

$$m = \frac{\Delta y}{S} \quad (9)$$

The first derivative of equation 3 give us the catenary tangent:

$$\frac{dy}{d\bar{x}} = \sinh\left(\frac{W\bar{x}}{H}\right) \quad (10)$$

\bar{x}_f is the point where the slope equals m , so we have:

$$\bar{x}_f = \frac{H}{W} \operatorname{asinh}\left(\frac{\Delta y}{S}\right) \quad (11)$$

The sag D is the vertical distance from the imaginary line that connects A and B , y_r , and the catenary y_c , both in \bar{x}_f

$$\begin{aligned} D &= y_r(\bar{x}_f) - y_c(\bar{x}_f) \\ y_r(\bar{x}_f) &= \frac{H}{W} \cosh\left(\frac{WS_B}{H}\right) - \frac{H}{W} - m(S_b - x_f) \\ y_c(\bar{x}_f) &= \frac{H}{W} \cosh\left(\frac{W\bar{x}_f}{H}\right) - \frac{H}{W} \end{aligned} \quad (12)$$

2.2.3 Computing the horizontal tension

The total length L of the cable can be computed from $L = L_A + L_B$

$$L_A = \frac{H}{W} \sinh\left(\frac{WS_A}{H}\right) \quad L_B = \frac{H}{W} \sinh\left(\frac{WS_B}{H}\right) \quad (13)$$

The cable longitudinal tension at a distance $S/2$ from O is

$$Ten = H \cosh\left(\frac{WS}{2H}\right) \quad (14)$$

Suppose two different cable states 0 y 1. There are now two longitudinal tensions Ten_0 and Ten_1 , two horizontal tensions H_0 and H_1 , two temperatures T_0 and T_1 and the two lengths L_0 and L_1 . Then we have the *state change equation*:

$$L_1 = L_0 \left[1 + a(T_1 - T_0) + \frac{Ten_1 - Ten_0}{EA} \right] \quad (15)$$

Where a is the coefficient of dilatation, E the elasticity module and A the section area. Notice that the new length must also satisfy equation 13. Usually numerical methods must be used to satisfy simultaneously equations 13 and 15.

Assuming that W does not change from state 0 to state 1, the value of H_1 is obtained by the solution of

$$\begin{aligned} &\frac{H_1}{W} \sinh\left(\frac{WS_A}{H_1}\right) + \frac{H_1}{W} \sinh\left(\frac{W(S-S_A)}{H_1}\right) = \\ &\left\{ \frac{H_0}{W} \sinh\left(\frac{WS_A}{H_0}\right) + \frac{H_0}{W} \sinh\left(\frac{W(S-S_A)}{H_0}\right) \right\} \\ &\left[1 + a(T_1 - T_0) \frac{1}{EA} \left[H_1 \cosh\left(\frac{WS}{2H_1}\right) - H_0 \cosh\left(\frac{WS}{2H_0}\right) \right] \right] \end{aligned} \quad (16)$$

3 Modelica implementation

Conductor and span parameters are stored in separated records named *ConductorData* and *SpanData*. Day of the year and time of the day are stored in a third record whose name is *TimeData*. Tables 1 to 3 summarize the parameters in each record.

3.1 Thermal model

14 functions have been defined for the implementation of the thermal model (see table 4). 3 classes have been also designed:

ConvectionHeatFlow: a model similar to the *HeatTransfer.Convection* model, but whose parameters are driven by the conductor, span and time parameters.

declaration	meaning
Real D	External diameter
Real a	Coefficient of dilatation
Real E	Module of elasticity
Real W	Linear weight
Real A	Cross section area
Real C	Linear heat capacity
Real R_ref	Linear electrical resistance
Real T_ref	Temperature of reference
Real alpha	Slope of resistance change
Real abs =0.5	Absorvity
Real emi =1.0	Emissivity

 Table 1: Parameters in the *ConductorData* record

declaration	meaning
Real He	Altitude above sea level in m
Real L	Latitude in deg
Real Zl	Azimuth of the line in deg
Real S	Span length en m
Real Dy	Support difference of level in m
Real T_0	Temperature of conductor in state of reference in K
Real Ten_0	Tension of conductor in state of reference in KgF
Real L_0	Lenght of conductor in state of reference in m

 Table 2: Parameters in the *SpanData* record

declaration	meaning
Integer Day	Day of the year (1-365)
Real Hour	Time of the day (0-24, 13.5 means 1:30 pm)

 Table 3: Parameters in the *TimeData* record

SolarHeatFlow: a model similar to the *HeatTransfer.PrescribedHeatFlow* model, but whose value is driven by sun and span position.

StandAloneHeatingResistor: a model similar to the *Electrical.Analog.Basic.HeatingResistor* model, that also computes the heat gain by Joule effect for a certain electrical current.

Conductor: it is a Heat Capacitor with a temperature signal port.

3.2 Mechanical and geometrical model

Mechanical and geometrical model is implemented by 4 functions (table 5) and 4 main classes:

CatenaryStateChange: this class is the implementation of the state change equations 15 and 16. (See File 1). This class is perhaps, the most important of the mechanical model. Here we establish that the state equation must also satisfy the new length condition of 13. In order to help a fast solution of the state change equations, a starting point for $\alpha = H/W$ can be set. We suggest to use 300².

Catenary: it is a partial class that joins the thermal, mechanical and geometrical models (see 2). In one hand it implements equation 1 as a *Conductor* (i.e. a heat capacitor) whose heat port has attached models for the heat flows (joule effect q_J , solar radiation q_s , convection q_c and radiation q_r); air temperature T_a is included as a prescribed temperature. In the other hand it has also a component of the *CatenaryStateChange* class; the temperature of the Conductor is used as an input for the State Change analysis, whose main output is the sag D calculation. In order to use this class, a derived class must be designed, so the electrical current I is defined. See File 3.

ElectricalCatenary: it is a derived class from *Catenary* class, in which a electrical current signal is attached to the conductor

²As an example, in the simulation shown in section 4.1 the numerical methods have found $\alpha = 394.7$ for $t = 0$

function	compute:
AirConductivity	thermal air conductivity as a function of film air temperature
AirDensity	air density as a function of altitude and film air temperature
AirViscosity	air viscosity as a function of film air temperature
AngleFactor	correction factor for convection heat losses as a function of the angle between the wind and conductor
Asinh	$asinh(x)$
ConvectionFlow	convection heat losses
ForcedConvectionHigh	forced convection heat losses for high wind speeds
ForcedConvectionLow	forced convection heat losses for low wind speeds
ForcedConvection	forced convection heat losses for any wind speed
NaturalConvection	natural convection heat losses
FilmTemperature	film air temperature as a function of air and conductor temperatures
SolarAltitude	sun Altitude as a function of latitude, day of the year and time of the day
SolarAzimuth	sun azimuth as a function of latitude, day of the year and time of the day
SolarFlux	solar heat gain as a function of altitude, solar altitude, sun and conductor azimuths and type of atmosphere

Table 4: Functions for the thermal model

function	compute:
CatenaryLenght	equation 13
CatenarySag	equation 12
CatenarySa	equation 8
CatenaryXbar	equation 3

Table 5: Functions for the mechanical and geometrical model

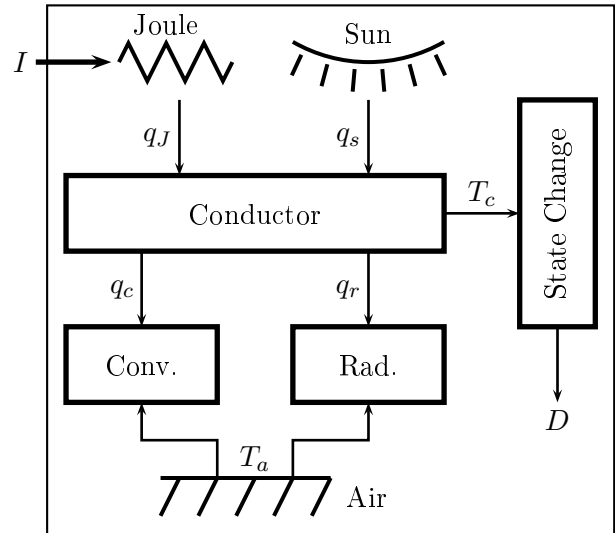


Figure 2: Catenary model

StandAloneCatenary: it is a derived class from *Catenary* class, the electrical current value is a Real, without attaching any signal to the conductor. In File 2 we show how to use this class for some specific simulation conditions.

4 System analysis and examples

In this section we illustrate the analysis capabilities of the model. To do it, we have applied the model to a real span of a 230KV distribution system in Bogotá, Colombia. The model was implemented using OpenModelica. As a part of a previous cargability study the span was modeled in detail ([5]). Most of the parameters were available (see table 6), however some experiments were conducted in order to identify the actual values of the following parameters:

- Emissivity.
- Absorvity.
- Conductor length in the state of reference.

Span parameters	
Conductor	Peacock
Altitude	2600 <i>msnm</i>
Latitude	4.779423° <i>Norte</i>
Azimuth	76.14°
Horizontal distance between supports	82.31 <i>m</i>
Difference of levels between supports	0.4; <i>m</i>
Nominal longitudinal tension	2970 <i>Kgf</i>
Conductor parameters	
External diameter	24.2 <i>mm</i>
Linear resistance at 25°C	$9.7 * 10^{-5}$ <i>ohm/m</i>
Linear resistance at 75°C	0.000116 <i>ohm/m</i>
Aluminium mass	0.79716 <i>Kg/m</i>
Steel mass	0.31227 <i>Kg/m</i>
Linear weight	1.16 <i>Kg/m</i>
Nominal elasticity module	0.7530 * 10^6 <i>KG/cm²</i>
Nominal coefficient of dilatation	$19.73 * 10^{-6}$ <i>1/°C</i>
Cross section area	3,4638 <i>cm²</i>

Table 6: Parameters for the examples

4.1 Heating analysis

First we study the change of conductor temperature for specific operation conditions. In this example we vary just two operation conditions through a 24 hour period: electrical current I (Figure 3) and air temperature T_a (Figure 4). These operation conditions may be considered as inputs for the present analysis where as the conductor temperature T shown in figure 5 is computed as an output.

4.2 Cargability analysis

Cargability analysis is the study of the maximum amount of electrical current that can pass through the conductor without getting a limit condition. In this example we state the limit condition as a conductor temperature of $75^\circ C$ and suppose the air temperature profile shown in figure 6. Cargability has been drawn in figure 7. Notice that the electrical current now is considered as an output of the model. The simulated model is shown in

File 4.

4.3 Sag analysis

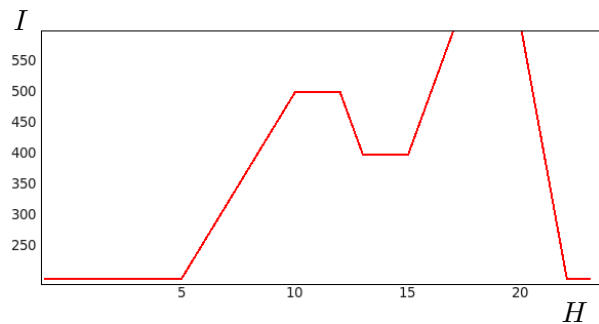
We now analyze the relation between Sag and three variables: electrical current I , air temperature T_a and conductor temperature T . We use the current profile of figure 3 and the air temperature profile of figure 6, and plot the sag D against the three variables (figures 8 to 10).

Notice that neither I nor T_a can explain alone the sag D . As the 24 hour cycles of I and T_a have different shapes (figures 3 and 6), and because of the nonlinear dynamic nature of the phenomena, during the 24 hour the same electrical current can occur two or more times with different sags.

Also notice that inspite the non linearities present in the equations of section 2.2, the relation between conductor temperature T and sag D is almost linear. This is why some facilities are now measuring the conductor temperature online, and not just the electrical current, in order to study the real online cargability.

4.4 Catenary analysis

We can also draw the catenary. To do that we use the dummy variable $x = St$ using $\dot{x} = S$, and use equation 3 in a simulation with stop time of 1s. An example is shown in figure 11


 Figure 3: Example 1. Electrical current I (A) vs Time of the day H (h)

5 Conclusions

A model for bare overhead conductors has been presented. It combines thermal, mechanical and geometrical phenomena. The core of the thermal model is a heat capacitor whose parameters and

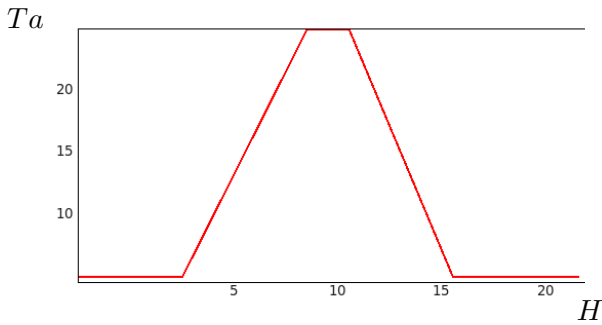


Figure 4: Example 1. Air temperature T_a (°C) vs Time of the day H (h)

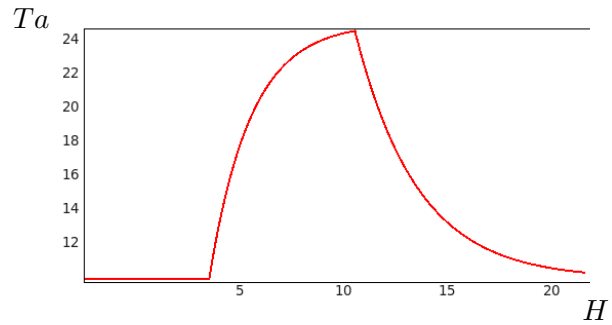


Figure 6: Example 2. Air temperature T_a (°C) vs Time of the day H (h)

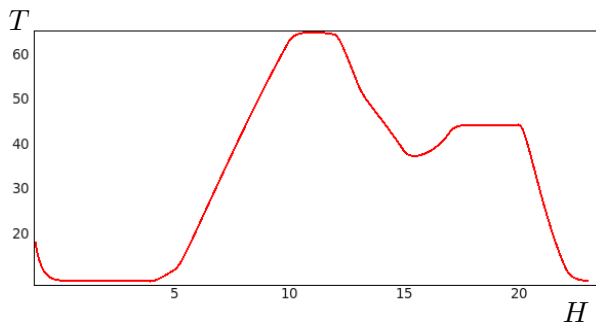


Figure 5: Example 1. Conductor temperature T (°C) vs Time of the day H (h)

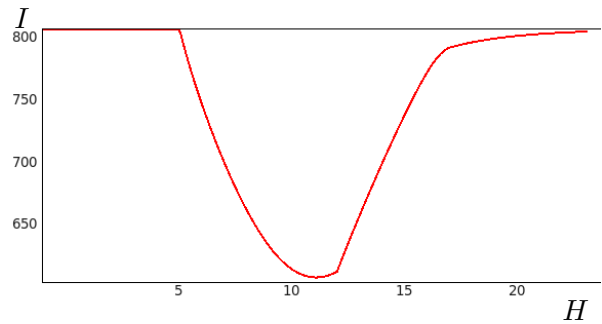


Figure 7: Example 2. Maximum electrical current I (A) vs Time of the day H (h)

inputs are driven by operation conditions as stated by the IEEE 738 standard. The mechanical model solves the state change equation for the conductor temperature of the thermal model and calculates the catenary geometry.

By using Modelica, two advantages have been found, that are clear examples of the object-oriented modeling advantages:

- The solution of the state equation is very simple. We just need to write two conditions for the conductor length. OpenModelica has generated the source code that we need in order to solve the equation.
- Cargability analysis is also simple. IEEE 738 standard give us a way to compute conductor temperature from an electrical current for specific operation conditions. However, the inverse problem (compute the current for a conductor temperature) is just solved for steady state in the standard. Using Modelica and OpenModelica the cargability problem is solved without any new equation.

We plan to create a Modelica library for bare overhead conductors in the short term. The library will include the parameters for the most common commercial conductors, and more analysis functionalities.

References

- [1] IEEE Power Engineering Society, IEEE Standard for Calculating the Current-Temperature of Bare Overhead Conductors IEEE Std 738-2006. January 2007.
- [2] OpenModelica, <http://www.openmodelica.org/> last visit: november 22, 2010.
- [3] Peter Fritzson, Peter Aronsson, Adrian Pop, Håkan Lundvall, Kaj Nyström, Levon Saldamli, David Broman, Anders Sandholm: OpenModelica - A Free Open-Source Environment for System Modeling, Simulation, and Teaching, IEEE International Symposium on Computer-Aided Control Systems Design, October 4-6, 2006, Munich, Germany

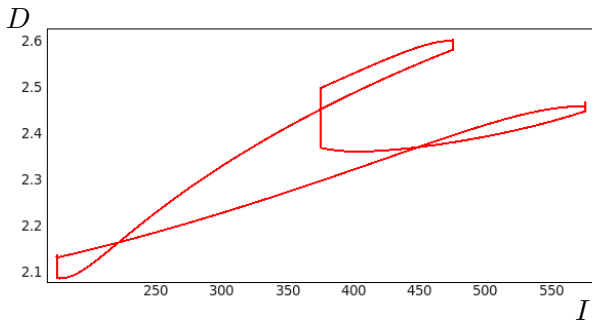


Figure 8: Example 3. Sag D (m) vs Electrical current I (A)

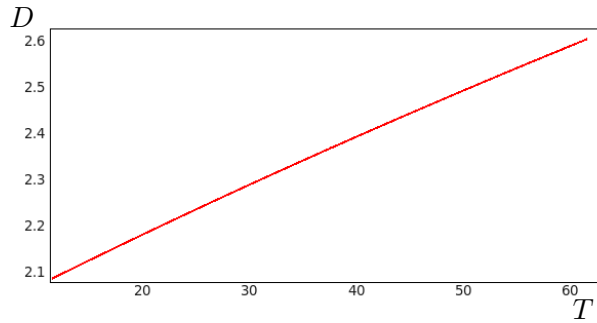


Figure 10: Example 3. Sag D (m) vs Conductor temperature T (°C)

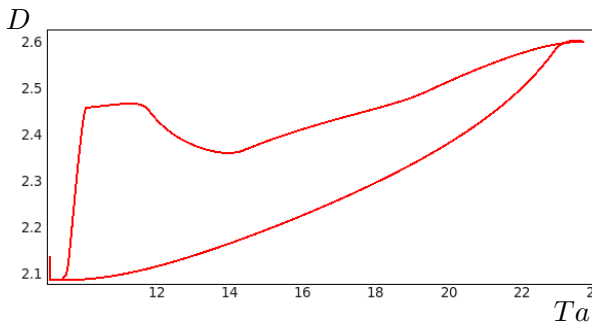


Figure 9: Example 3. Sag D (m) vs Air temperature T (°C)

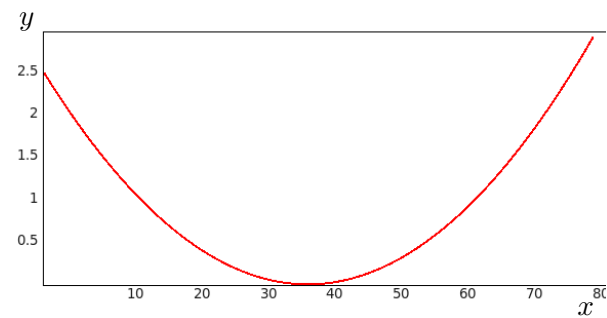


Figure 11: Example 4. Shape of the catenary. Height y (m) vs Horizontal distance x (m)

- [4] Donald G. Fink, H. Wayne Beaty. Standard Handbook for Electrical Engineers. McGraw-Hill Professional
- [5] Oscar Duarte, Jaime Alemán, René Soto, Estrella Parra, Francisco Amórtegui, Wilson Aldana. Identificación de parámetros y estudio de cargabilidad en algunas Líneas de Transmisión de Codensa. Technical report. 2009.

File 1: CatenaryStateChange.mo

```

within Catenary;
model CatenaryStateChange
  parameter Real W;
  parameter Real S;
  parameter Real Dy;
  parameter Real T_0;
  parameter Real Ten_0;
  parameter Real L_0;
  parameter Real a;
  parameter Real E;
  parameter Real A;
  Real T, Ten, L, H, alpha ( start =300 ), Sag;
  TemperatureSignalPort port_T;
equation
  T=port_T.T;
  alpha=H/W;
  Ten=H*cosh ( S/2/alpha );
  L=L_0*(1 + a*(T-T_0) + (Ten-Ten_0)/(E*A)
  );
  L=CatenaryLenght ( S, alpha ,Dy );
  Sag=CatenarySag ( S, alpha ,Dy );
end CatenaryStateChange;

```

File 2: MyStandAloneLine.mo

```

within Catenary;

model MyStandAloneLine
  parameter Real Imax=500;
  parameter Real Imin=200;
  parameter Real Imin2=400;
  parameter Real Imax2=600;
  parameter Real currentTable[:,2]={
    {0,Imin},
    {60*60*6,Imin},
    {60*60*11,Imax},
    {60*60*13,Imax},
    {60*60*14,Imin2},
    {60*60*16,Imin2},
    {60*60*18,Imax2},
    {60*60*21,Imax2},
    {60*60*23,Imin},
    {60*60*24,Imin}};
  parameter Real Tmax=25;
  parameter Real Tmin=5;
  parameter Real airTempTable[:,2]={
    {0,273.15+Tmin},
    {60*60*5,273.15+Tmin},
    {60*60*11,273.15+Tmax},
    {60*60*13,273.15+Tmax},
    {60*60*18,273.15+Tmin},
    {60*60*24,273.15+Tmin}};
  extends StandAloneCatenary(I(start=Imin)
  );
  Modelica.Blocks.Sources.TimeTable
    current_source(table=currentTable);
  Modelica.Blocks.Sources.Constant
    windVel(k=0.61);
  Modelica.Blocks.Sources.Constant
    windDir(k=0);
  // Modelica.Blocks.Sources.Constant
    airTemp(k=273.15+20);
  Modelica.Blocks.Sources.TimeTable
    airTemp(table=airTempTable);
  Modelica.Blocks.Sources.BooleanConstant
    atmos(k=true);
equation
  I=current_source.y;
  windVelocity=windVel.y;
  windDirection=windDir.y;
  atm=atmos.y;
  ta=airTemp.y;
end MyStandAloneLine;

```

File 3: Catenary.mo

```

within Catenary;

partial class Catenary
  parameter ConductorData con(D=24.2,a
    =19.7e-6,E=0.753e6,A=3.4638,W=1.16,C
    =909.9,R_ref=0.000097,T_ref
    =273.15+25,alpha=3.8e-7,abs=0.5,emi
    =1.0);
  parameter SpanData span(He=2600,L
    =4.779420,Zl=76.14,S=82.31,Dy=0.4,T_0
    =273.15+20,Ten_0=2970,L_0=82.54);
  parameter TimeData to(Hour=0,Day=57);
  parameter Real InitialTemp=20;
  //Weather
  Real ta,taCelcius,windVelocity,
    windDirection;
  Boolean atm;
  // Thermal
  Conductor Wire(C=con.C);
  Modelica.Thermal.HeatTransfer.Sources.
    PrescribedTemperature Env;
  Modelica.Thermal.HeatTransfer.Components
    .BodyRadiation Rad(Gr=con.emi*con.D
    *0.0178/5.6704);
  SolarHeatFlow Sun(He=span.He,L=span.L,
    absorvity=con.abs,Zl=span.Zl,area=con
    .D/1000,Hour=to.Hour,Day=to.Day);
  ConvectionHeatFlow Conv(axis=true,D=con.
    D,He=span.He);
  //Mechanical & geometric
  CatenaryStateChange Sag(a=con.a,E=con.E,
    A=con.A,W=con.W,S=span.S,Dy=span.Dy,
    T_0=span.T_0,Ten_0=span.Ten_0,L_0=
    span.L_0);
  Real T(start=InitialTemp,fixed=false),Qj
    ,Qs,Qr,Qc,hour,sag;
equation
  hour=time/(60*60);
  T=Wire.T-273.15;
  Qj=-HR.heatPort.Q_flow;
  Qc=Conv.Q_flow;
  Qs=Sun.Q_flow;
  Qr=Rad.Q_flow;
  sag=Sag.Sag;
  // weather signals to components
  Conv.Vw=windVelocity;
  Conv.phi=windDirection;
  Sun.Atm=atm;
  Env.T=ta;
  taCelcius=ta-273.15;
  //Thermal
  connect(HR.heatPort,Wire.port);
  connect(Wire.port,Sun.port);
  connect(Wire.port,Rad.port_a);
  connect(Rad.port_b,Env.port);
  connect(Wire.port,Conv.solid);
  connect(Conv.fluid,Env.port);
  //Mechanical and geometric
  connect(Wire.port_T,Sag.port_T);
end Catenary;

```

File 4: Cargability.mo

```

within Catenary;

model Cargability
  parameter Real TAmin=10;
  parameter Real TAmx=25;
  parameter Real TCmax=75;
  parameter Real Vvto=0.61;
  parameter Integer TAFlag=3;
  extends StandAloneCatenary; /* (
    ta Celcius (start=TAmin),
    ta (start=273.15+TAmin),
    Wire.T(start=273.15+TCmax)); */
  Modelica.Blocks.Sources.Constant
    temperature_source(k=273.15+TCmax);
    // Ex. of
    Temperature of conductor known
  Modelica.Blocks.Sources.Constant
    windVel(k=Vvto);
  Modelica.Blocks.Sources.Constant
    windDir(k=0);
  Modelica.Blocks.Sources.Trapezoid
    airTemp1(offset=273.15+TAmin,
    amplitude=TAmx-TAmin, start Time
    =60*60*6, rising=60*60*5, width
    =60*60*2, falling=60*60*5, period
    =60*60*24);
  Modelica.Blocks.Sources.Sine
    airTemp2(offset=273.15+(
    TAmx+TAmin)/2, amplitude=(TAmx-TAmin
    )/2, freqHz=1/(60*60*24), phase=-
    Modelica.Constants.pi/2);
  Modelica.Blocks.Sources.Exponentials
    airTemp3(offset=273.15+TAmin,
    outMax=TAmx-TAmin, start Time=60*60*6,
    riseTime=60*60*7, riseTimeConst
    =60*60*2, fallTimeConst=60*60*3);
  Modelica.Blocks.Sources.BooleanConstant
    atmos(k=true);

equation
  Wire.T=temperature_source.y;
  windVelocity=windVel.y;
  windDirection=windDir.y;
  atm=atmos.y;
  if TAFlag==1 then
    ta = airTemp1.y;
  elseif TAFlag==2 then
    ta = airTemp2.y;
  elseif TAFlag==3 then
    ta = airTemp3.y;
  end if;
end Cargability;

```

File 5: SagAnalysis.mo

```

within Catenary;

model SagAnalysis
  parameter Real TAmin=10;
  parameter Real TAmx=25;
  parameter Real Imax=500;
  parameter Real Imin=200;
  parameter Real Imin2=400;
  parameter Real Imax2=600;
  parameter Real mitable[: ,2]={
    {0, Imin},
    {60*60*6, Imin},
    {60*60*11, Imax},
    {60*60*13, Imax},
    {60*60*14, Imin2},
    {60*60*16, Imin2},
    {60*60*18, Imax2},
    {60*60*21, Imax2},
    {60*60*23, Imin},
    {60*60*24, Imin}};
  extends StandAloneCatenary(T(start=60));
  Modelica.Blocks.Sources.TimeTable
    current_source(table=mitable);
  Modelica.Blocks.Sources.Constant
    windVel(k=0.61);
  Modelica.Blocks.Sources.Constant
    windDir(k=0);
  // Modelica.Blocks.Sources.Constant
  airTemp(k=273.15+20);
  Modelica.Blocks.Sources.Exponentials
    airTemp(offset=273.15+TAmin,
    outMax=TAmx-TAmin, start Time=60*60*6,
    riseTime=60*60*7, riseTimeConst
    =60*60*2, fallTimeConst=60*60*3);
  Modelica.Blocks.Sources.BooleanConstant
    atmos(k=true);

equation
  I=current_source.y;
  windVelocity=windVel.y;
  windDirection=windDir.y;
  atm=atmos.y;
  ta=airTemp.y;
end SagAnalysis;

```

Modeling and Simulation of AMT with MWorks

Ming Jiang, Jiangang Zhou, Wei Chen, Yunqing Zhang, Liping Chen
CAD Center, Huazhong University of Science and Technology, China
zhangyq@hust.edu.cn

Abstract

This paper presents a detailed AMT model composed of various components from multi-domains like mechanical systems (clutch, gear pair, synchronizer, etc.), pneumatic actuator systems (clutch actuation system, gear select actuation system, gear shift actuation system, etc.). The model is implemented using the Modelica modeling language in an object-oriented environment. The Modeling and simulation of the AMT model was carried out on MWorks, which is developed by Huazhong University of Science and Technology.

Keywords: modeling; simulation; automatic mechanical transmission (AMT); multi-domain;

1 Introduction

The automatic mechanical transmission (AMT) is generally constituted by a dry clutch and a multi-speed gearbox, both equipped with hydraulic, pneumatic or electric actuators, which are driven by an Electronic Control Unit (ECU). Compared to manual transmission, the AMT allows to improve driving comfort and shift quality, and can get better fuel economy. Compared to automatic transmission (AT), The AMT has the advantage of lower weight and higher efficiency [1]. And moreover, the product line of manual transmission can be reused for AMT. So the AMT has attracted increasing interesting from automotive researches, and the vehicles equipped AMT are spreading in recent years, especially in Europe, Japan and China.

The AMT always uses electronic sensors, processors and hydraulic, pneumatic or electric actuators to execute clutch actions and gear shifts on the command of the driver [2, 3]. The clutch and gearbox are controlled by electronic computers and hydraulic, pneumatic or electric actuators. The AMT operates the clutch and throttle to match velocity according to the control logic. Such systems coupled with various physical domains have great influence on the dynamic behavior of the vehicle, such as shift quality, dri-

veability, fuel economy, acceleration, etc.[4]. Many researches have been carried out to study the dynamic performance and control logic of the AMT systems. However, many previous works considered the AMT actuation system as an ideal system and neglected the transient behavior of the clutch actuation gear box actuation due to time delays, non-linear dynamic characteristic, external disturbance and parameter uncertainty of the pneumatic, hydraulic, mechanic and electric components [5, 6]. The simplification of the clutch and gear box system can't represent a real AMT. The actuator dynamics cannot be neglected at all, since they can affect the clutch action and shifting performances. So some researchers built more detailed model to study the AMT system. Lucente [7] described detailed models of clutch electro-hydraulic and electro-mechanical actuators for AMT. Zhao [8] presented the modeling and simulation of clutch actuator, and addressed the analysis on the DC permanent magnet motor (power source). This paper presents a detailed an AMT model composed of various components from multi-domains like mechanical systems (clutch, gear pair, synchronizer, etc.), pneumatic actuation systems (clutch actuation system, gear select actuation system, gear shift actuation system, etc.).

The model is implemented using the Modelica modeling language, which is a non-proprietary, object-oriented, equation based language to conveniently model complex physical systems containing, e.g., mechanical, electrical, electronic, hydraulic, thermal, control, electric power or process-oriented subcomponents [9]. The simulation was carried out by MWorks [10], which is developed by Huazhong University of Science and Technology.

The paper is organized as the follows. Section 2 introduces the modeling for clutch, including clutch plate, diaphragm spring, clutch lever and clutch damper, etc. Section 3 introduces the modeling for gear box, such as gear pairs, synchronizer, etc. The clutch actuator and gear actuator including pneumatic cylinder and valves, etc. are discussed in section 4. The simulation results and discussions are illustrated in section 5. Section 6 offers our conclusions and further study.

2 Clutch Model

The function of an engaging friction clutch is to transmit torque gradually, to avoid high accelerations or jerks, when the engine is connected to the rest of the driveline. As can be seen in Fig.1, the clutch system consists of clutch plate, diaphragm, throw-out bearing, pressure plate, etc. The flywheel is connected to the engine, the clutch plate is connected to the transmission, and the clutch lever is connected to pneumatic clutch actuation system, which will be described in the following. When clutch actuator acted on the clutch lever, the throw-out bearing pushed the diaphragm spring in sequence, then the diaphragm spring pull the pressure plate apart away the clutch disc, which in turn release presses against the flywheel. So there is no torque transmit from the flywheel to the transmissions. And when the clutch actuation released, the diaphragm spring push the pressure plate against the clutch disc, which in turn presses against the clutch plate. This locks the engine to the transmission input shaft, making them spin at the same speed.

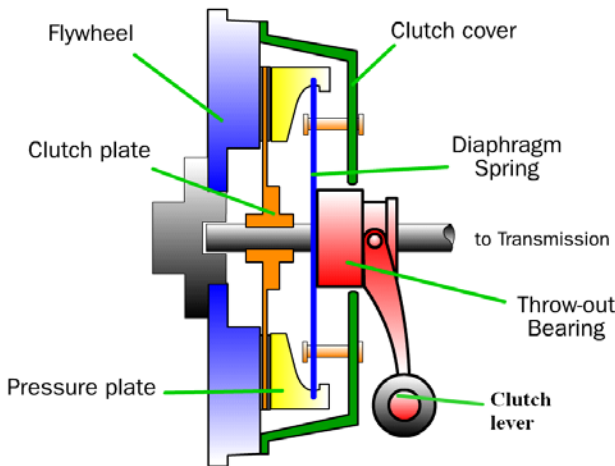


Fig. 1 The clutch system

2.1 Clutch Plate

According the function of the clutch, there are three distinct modes of operation of clutch plate: free, where the two plates transmit no torque; slipping, where the two plates have differing angular velocities; and lockup, where the two plates rotate together. The clutch system was analyzed using a lumped-parameter model.

The clutch plate model was developed based on clutch model included in Modelica standard library. The input signal u was designed as a normalized form in the standard library. In this paper, the normal force was designed as input signal.

The torque capacity of the clutch is a function of its size, friction characteristics, and the normal force that is applied [8].

$$T_{f \max} = \mu F_n n \left(\frac{2}{3} \right) \left(\frac{R_o^3 - R_i^3}{R_o^2 - R_i^2} \right) \quad (1)$$

Where μ is coefficient of friction that can be formulated as a function of clutch slip, F_n is normal force on clutch face that depends on the apply pressure, R_o is outside radius of friction disc, R_i is inside radius of friction disc, n is the number of friction discs.

2.2 Diaphragm Spring

The clutch plates are pushed together by diaphragm spring. However, it is difficult to give the equation of the diaphragm. In this paper, a lookup table based on experiment data is used to simulate the mechanical character diaphragm. A linear interpolation to the characteristic curve is used to model the diaphragm spring. The normal actuation force on the clutch plate through the model of diaphragm is acquired by the diaphragm model.

2.3 Clutch Lever

The clutch lever is modeled by lever principle, and described as follows:

$$F_A = i * F_B \quad (2)$$

$$S_A = S_B / i \quad (3)$$

Where, F_A and F_B are the force acted on the two sides, and S_A and S_B are the displacement of the two sides, and i is the lever ratio.

2.4 Clutch damper

The clutch damper reduces the torque and vibration caused by engaging the clutch pedal's effect on the rest of the engine system. It is modeled as parallel torsional spring and damper. It is described as following equation:

$$T_d = k * \theta + d * w \quad (4)$$

Where, T_d is the torque transmitted by the clutch damper, k and d is the stiffness and damper of the clutch damper, respectively, θ and w is the relative angle and relative rotational speed of the two plates, respectively.

2.5 Clutch Model Assembly

The clutch library includes clutch components, such as lever, release bearing, clutch damper, diaphragm, clutch plate, etc., which is can be seen in Fig. 2.

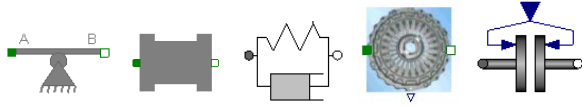


Fig. 2 The clutch library

According to the subsystem model of clutch system mentioned above, a clutch model assembly was built by graphic user interface, as can be seen in Fig.3. The model includes clutch lever, throw-out bearing, diaphragm spring, clutch plate, clutch damper, etc. Where, the port A is connected to the engine, the port B is connected to the input shaft of the gearbox, and the port C is connected to the clutch actuation system.

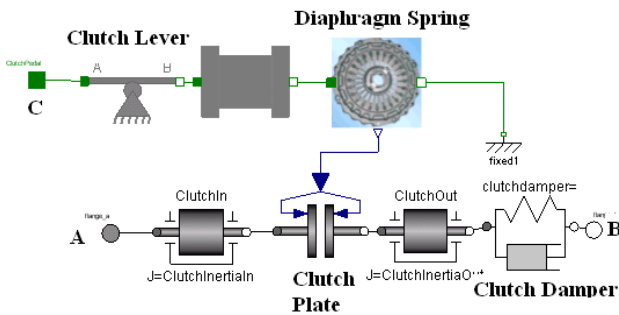


Fig.3 The clutch model

3 GearBox Model

The gearbox provides speed and torque conversions from a rotating power source to another device using gear ratios. The gearbox presented in this paper was equipped on heavy truck with 14 speeds. The gearbox included front sub-gearbox assembly, main gearbox assembly, and rear sub-gearbox assembly, which is planetary gearbox. The gear model, planetary gearbox model and synchronizer model will be introduced in the following.

3.1 Gear Model

A gear is a rotating machine part having cut teeth, or cogs, which mesh with another toothed part in order to transmit torque. The gear model is described as:

$$T_{in} = T_{out} + r * (F_1 + F_2) \tag{5}$$

$$\omega_{in} = \omega_{out} \tag{6}$$

$$v_1 = v_2 = r * \omega_{in} \tag{7}$$

Where, T_{in} is the torque acted on the input shaft, T_{out} is the output torque, ω_{in} and ω_{out} are the rotary speed of the gear, F_1 and F_2 are the force acted on the gear teeth, v_1 and v_2 are the velocity of the gear teeth, and r is the gear radius.

The gear library can be seen in Fig.4.

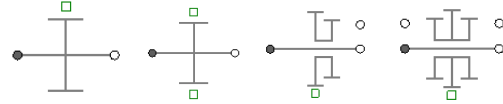


Fig. 4 The gear library

3.2 Planetary Gearbox

The planetary gear box is an ideal gear without inertia, elasticity, damping or backlash consisting of an inner sun wheel, an outer ring wheel and a planet wheel located between sun and ring wheel. The bearing of the planet wheel shaft is fixed in the planet carrier. The component can be connected to other elements at the sun, ring and/or carrier flanges. The planetary gearbox is described as follows:

The resulting torque on the ring gear is:

$$T_1 = \frac{r_2}{r_2 - r_1} * T_3 \tag{8}$$

The rotary velocity of the planet carrier is:

$$\omega_3 = \frac{r_2}{r_2 - r_1} * \omega_1 + \frac{r_1}{r_2 - r_1} * \omega_2 \tag{9}$$

The resulting torque on the planet gear is:

$$T_2 = \frac{r_1}{r_2 - r_1} * T_3 \tag{10}$$

Where, T_1 , T_2 and T_3 is torque on the ring gear, the planet carrier and the planet gear, respectively, ω_1 , ω_2 and ω_3 is rotary velocity of the ring gear, the planet carrier and the planet gear, respectively, r_1 is the radius of the planet gear, r_2 is the radius of the ring gear. The planetary gear can be seen in Fig.5.

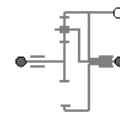


Fig. 5 The planetary gear

3.3 Synchronizer Model

The synchronizer is the most important shifting component in a gearbox. The functions of a synchronizer are: making disappear the angular velocity difference between the synchro hub and the gear to be engaged, impeaching the engagement of the gear

while the angular velocity difference exists, and connecting hub and gear, by gear splines, via sleeve, in order to allow power transmission when the gear changing is realized [11].

However, the dynamical behaviour of these three mechanical subsystems is rather complicated to simulate because of the large number of elements. It is not easy to study the entire gear-changing process in detail [12]. So some simplifications were taken on the synchronizer model. In this paper, the synchronizer model was considered as a clutch plate model mentioned above. And the transmission torque is:

$$M = \frac{\mu F_n r_m}{\sin \alpha} \quad (11)$$

Where μ friction coefficient between conical surfaces of the ring and the gear, α is the taper angle of the cone ring, r_m is mean radius of the cone, F is the gearshift force calculated by the gear actuator model. More details about the synchronizer can be referred to paper [12, 13].

The synchronizer library includes thrust piece, gear mesh, bumper stop, friction cone, etc., which is can be seen in Fig.6. The synchronizer model was shown in Fig.7.

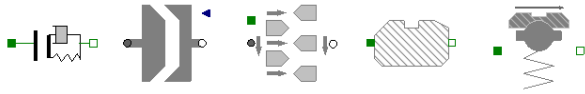


Fig. 6 The synchronizer library

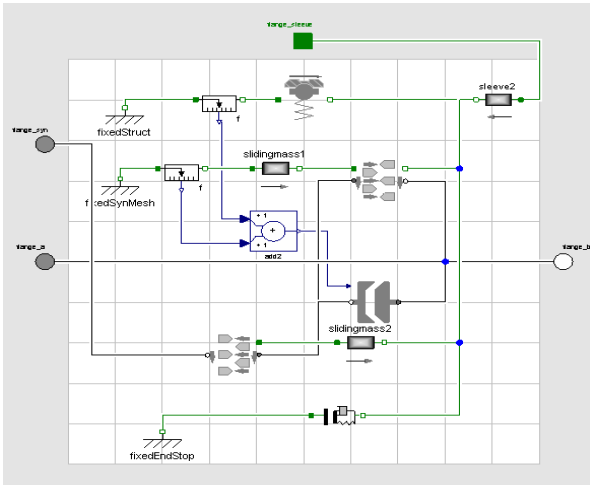


Fig. 7 The synchronizer model

3.4 Gearbox Assembly Model

The gearbox presented in this paper was a 14-speed gearbox. According to the subsystem model mentioned above, a gear box model assembly was built by the structure of the gearbox, as can be seen in Fig.8. The model includes gears, synchronizers, pla-

netary gearbox, gear lever, gear efficiency, etc. Where, the port A is connected to the clutch, the port B is connected to the output shaft of the gearbox, and the port C is connected to the gear actuator system, such as gear shift actuator, gear select actuator.

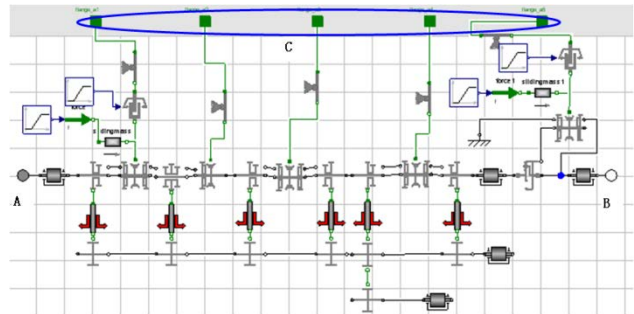


Fig.8 The gearbox assembly model

4 Actuator Model

The pneumatic actuator of AMT is consists of pneumatic source, pneumatic actuator of clutch, gear select actuator, gear shift actuator, and other components. The clutch actuator was controlled by the clutch control logic to exert a displacement on the clutch lever, and make the clutch engaged or disengaged by the control logic. The gear select actuator and shift actuator were connected to the gear box, and exert a force on the gear box to select or shift the gears according by the gearbox control logic. The pipe is neglected because it is very short.

The flow chart of the pneumatic actuator can be seen in Fig.9. Where, the reservoir is the air source of the pneumatic actuation system, the clutch cylinder is connected to the clutch lever, the gear shift cylinder, the gear select cylinder, the front and rear sub-gearbox cylinder were connected to the gearbox. The cylinder was controlled by the directional control valves to move to certain position with certain velocity. The valves were get signals by the control logic.

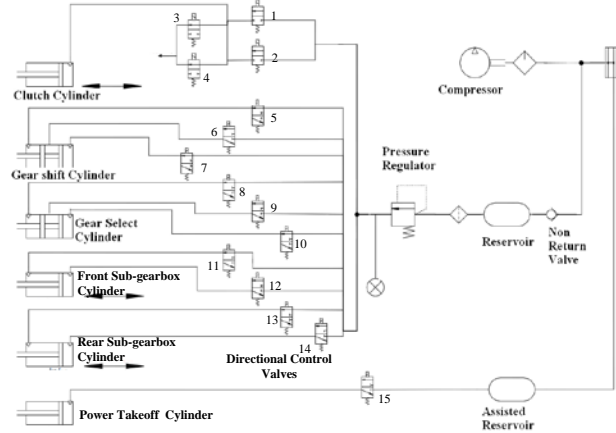


Fig.9 The Flow chart of the pneumatic actuator

5 Simulation and Results

To analysis the performance the AMT systems, the simulation about the clutch actuation system, the synchronizer and overall vehicle equipped AMT assembly were carried out in the following.

5.1 Synchronizer Simulation

In this model, the synchronizer body and sleeve have an initial rotational speed of 1500 rpm and the idle gear has an initial speed of 1000 rpm. The sleeve moves to the teeth of the synchronizer ring, the synchronizer body and sleeve synchronize with idle gear, the sleeve moves further until it meets the teeth to the idler gear and finally the idler gear is positively locked.

Fig. 10-12 shows the work phase of the synchronizer. When the synchronize ring was shifted, the normal force on the cone ring will increase, then down to zero, and the synchronizer ring will hold at the gear-mesh position for some time before the two side achieved the same speed.

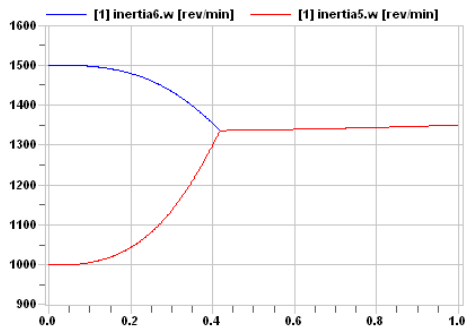


Fig.10 Rotational speed of two sides

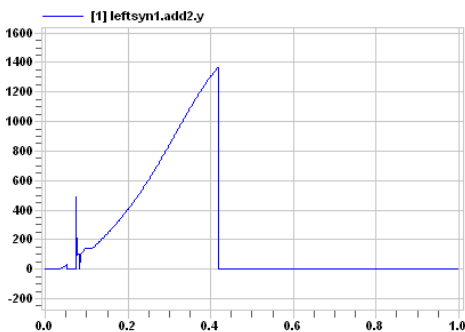


Fig.11 Normal force acted on the cone ring

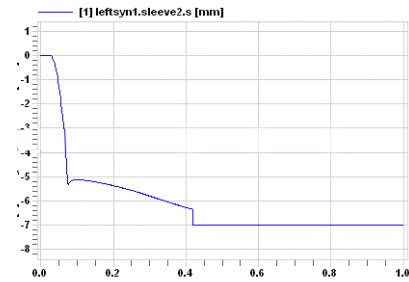


Fig.12 displacement of the synchronizer ring

5.2 Clutch System Actuation Simulation

This model tested the clutch response when the clutch actuator pushed the clutch lever, and disengage the clutch plate. The model can be seen in Fig.13. Fig.14 shows the control signal of clutch actuator. Fig.15-17 shows the performance of the clutch. The red line was the simulation results, and the blue line is the experiment results. The simulation results fit the experiment results very well.

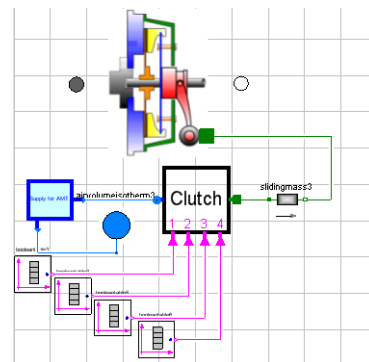


Fig.13 Clutch System Actuation Simulation

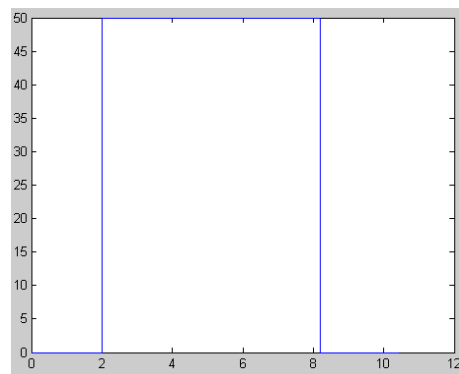


Fig.14 Input signal of clutch actuator

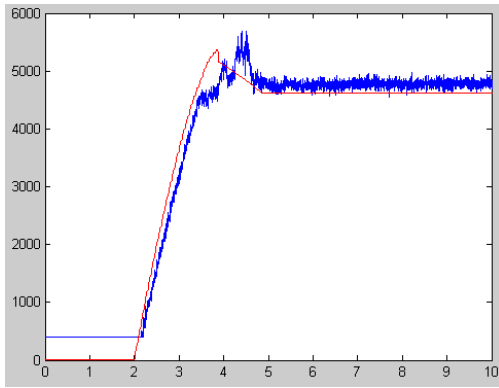


Fig.15 The displacement on the clutch lever

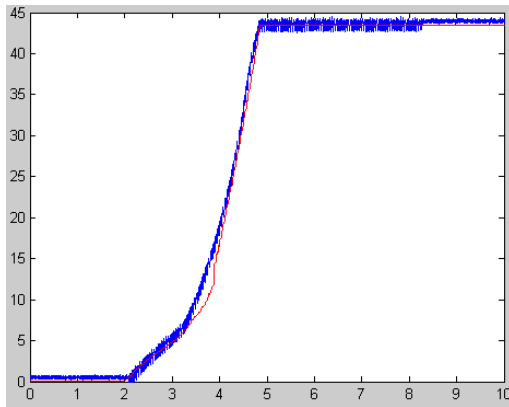


Fig.16 The Force on the clutch lever

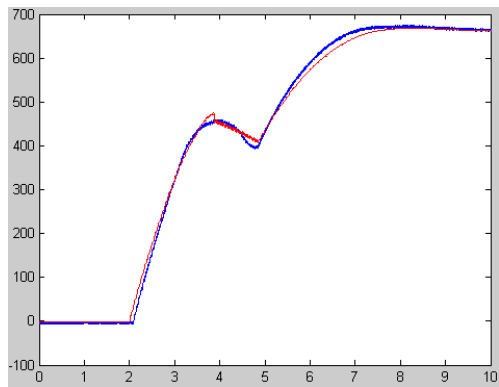


Fig.17 The pressure in the cylinder

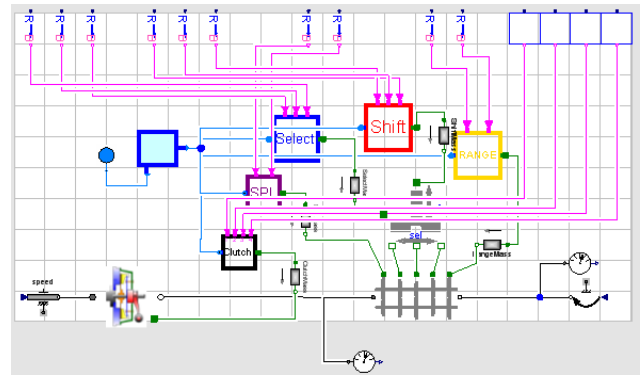


Fig.18 The pressure in the cylinder



Fig.19 The s-function model

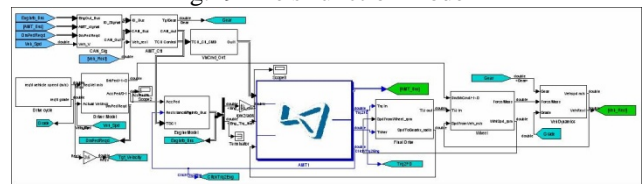


Fig.20 The overall vehicle model

5.3 Co-simulation with Vehicle Model

According to the topology of the AMT, the pneumatic actuator, clutch, gear box model were assembled together to get the AMT model, as can be seen in Fig.18.

The vehicle model was developed with Matlab/Simulink. To connect the AMT model with the overall vehicle model, the modelica model must be translated to simulink model. MWorks can compile the modelica model to s-function model, seen in Fig.19.

The input signal to AMT model include the clutch command, select gear command, shift gear command, front sub-gearbox shift command, rear sub-gearbox shift command, engine speed and torque, etc. The output signal to the vehicle model and control logic include the clutch cylinder position, the gear select actuator position, the gear shift actuator position, the front sub-gearbox shift actuator position, the rear sub-gearbox shift actuator position, output speed of the gearbox, etc. The overall vehicle model can be seen in Fig.20. The simulation was carried out to follow a given speed.

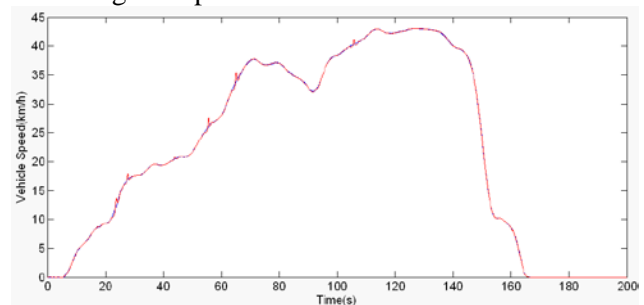


Fig.21 The vehicle speed

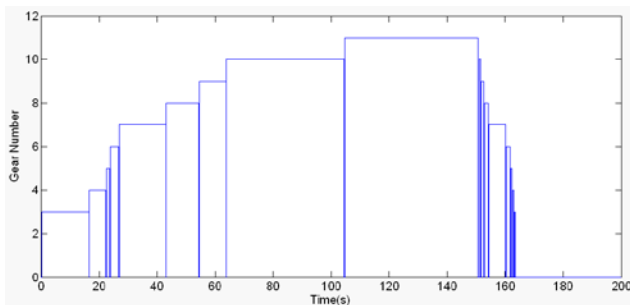


Fig.22 Gear number

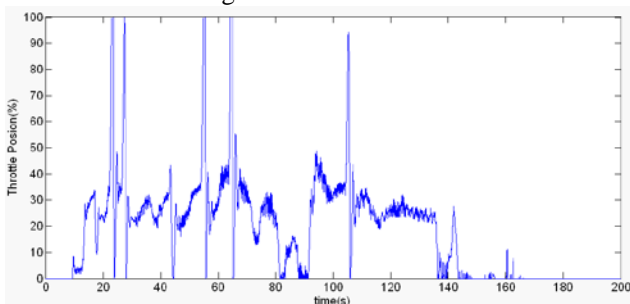


Fig.23 Throttle position

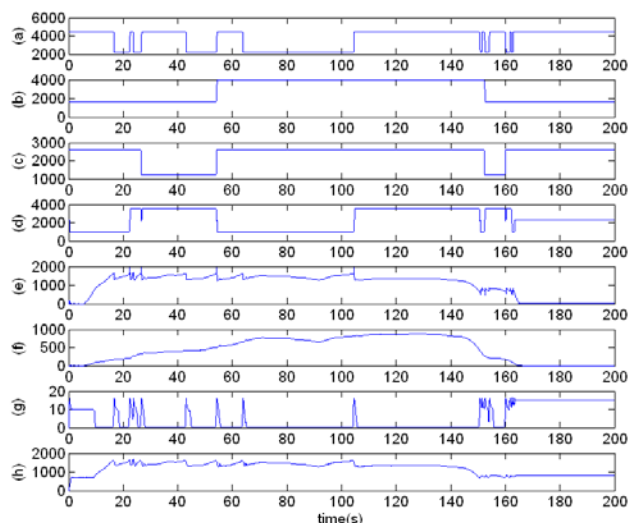


Fig.24 The performance of AMT

Fig. 21 shows the required vehicle speed and the actual vehicle speed. The red line is the required vehicle speed and the blue line is the simulation vehicle speed. Fig.22 shows the gear number, and Fig.23 shows the throttle position. Fig.24 shows the performance of the AMT model. Fig. a is the front sub-gearbox shift actuator position, Fig. b is the rear sub-gearbox shift actuator position, Fig. c is the gear select actuator position, Fig. d is the gear shift actuator position, Fig. e is the input speed of the gearbox, Fig. f is the output speed of the gearbox, Fig. g is the clutch cylinder position, and Fig. h is the engine speed.

6 Conclusions and Further Study

The paper addressed the problem of modeling AMT system. A detailed model of the driveline of a heavy truck with automated mechanical transmission was been developed. It considers both the dynamics of the transmission shafts, and the servo-actuated clutch and gearbox. Detailed models of the pneumatic actuators and its effects on driveline behavior and on performance during gear shift can be evaluated. The simulation results show that the detailed model can be used to estimate the influence of the clutch and gearbox actuation systems on the vehicle performance. The influence affected by the actuator on the dynamic behavior of the vehicle, such as shift quality, driveability, fuel economy, acceleration, etc. will be researched in the future.

Acknowledgement

This work was supported by the National High-Tech R&D Program, China (No. 2009AA044501).

References

1. Hiroshi K., Naoyuki O. Takashi O. and Masaru Y. Next-generation Fuel-efficient Automated Manual Transmission. *Hitachi Review*, 2004, Vol. 53, No. 4.
2. Florêncio D. and Assis E. The Manual Transmission Automated – Gearshift Quality Comparison to a Similar Manual System. SAE paper 2004-01-3363.
3. Turner A. J. and Ramsay K. Review and Development of Electromechanical Actuators for Improved Transmission Control and Efficiency. SAE paper 2004-01-1322.
4. Franceso V., Luigi I., Adolfo S. and Maurizio T. Modeling Torque Transmissibility for Automotive Dry Clutch Engagement. 2008 American Control Conference, Washington, USA, June 11-13, 2008
5. Zhao Y., Chen L., Zhang Y. and Yang J. Enhanced Fuzzy Sliding Mode Controller for Automated Clutch of AMT Vehicle. SAE paper 2006-01-1488.
6. Liu F., Li Y., Zhang, J., Huang H. and Zhao H. Robust Control for Automated Clutch of AMT Vehicle. SAE paper 2002-01-0933.
7. Lucente G., Montanari M., Rossi C. Modelling of an automated manual transmission system. *Mechatronics*, 2007, No. 17, pp.73–91.
8. Zhao L., Zhou Y. and Zheng L. Modeling and Simulation of AMT Clutch Actuator Based on SimulationX. *Computational Intelligence and Software Engineering*, 2009, pp.1-5.
9. Fritzson P., Vadim V. Modelica -- A Unified Object-Oriented Language for System Modeling and Simula-

- tion. Proceedings of the 12th European Conference on Object-Oriented Programming, 1998, pp.67 – 90.
10. Zhou F, Chen L. and Wu Yi., etc. MWorks: a Modern IDE for Modeling and Simulation of Multidomain Physical Systems Based on Modelica. Modelica 2006, September 4th – 5th, pp. 725-732.
 11. Manish K., Taehyun S., Zhang Y. Shift dynamics and control of dual-clutch transmissions. Mechanism and Machine Theory, 2007, Vol.42, pp.168–182
 12. Lovas L. and Play D. etc. Modelling of gear changing behavior. Periodica Polytechnica Ser. Transp. Eng. 2006, Vol. 34, No. 1–2, pp. 35–58.
 13. Lovas L., Play D., Marialigeti J. and Rigal J. F. Mechanical behaviour simulation for synchromesh mechanism improvements. Proceedings of the Institution of Mechanical Engineers, Journal of Automobile Engineering, 2006, Vol. 220, No. D7, pp. 919-945.

Variability and Type Calculation for Equation of Modelica model

Junjie Tang Jianwan Ding Liping Chen Xiong Gong
 National CAD Center
 1037 Luoyu Road, Wuhan, China

tjj.hust@gmail.com dingjw@hustcad.com Chenlp@hustcad.com Gongx@hustcad.com

Abstract:

Differential algebraic equations (DAEs), translated from Modelica model, is usually represented by bipartite graph. One of basic premises of creating bipartite graph is to determine types of variables and equations. Type calculation of Modelica equation has been researched and a serial of rules for variability and type calculation has been concluded in this paper.

Equation type is the type of variable that equation can solve. Equation type is calculated in symbolic by both variability and basic type of its sub-expressions. Generally, type calculation is a bottom-up way as expression is represented in form of tree. But, there are kinds of particular expressions, such as `integer()`, `noEvent()`, multi-output function call expression, etc, which may cause type and variability incompatible problem. The issue is discussed in the paper, and several rules for variability and type calculation are present. These rules will helps to debug out obscure errors, and several typical examples are present to show how the rules work.

Keyword: equation type; equation variability; compatibility of variability and type; model debug

1. Introduction

Differential algebraic equations (DAEs), generated by compiling and translating Modelica model, should be debugged by method of structural analysis [1], and be reduced and decomposed to subsystem serials [2], to reduce the scale of equation system and improve efficiency of numerical calculation. Most of these symbolic operations are usually processed based on representation of bipartite graph of DAEs, and one of basic premises of creating bipartite graph is determining type of variables and equations. The type of variable, obviously, is as defined in model, while the type of equation has to be deduced from its sub-expressions [3].

Equation type is the type of variable that equation can solve. Equation type is symbolic calculated by both variability and basic type of its sub-expressions. Generally, type calculation is a bottom-up way as expression is represented in form of tree. But, there are some particular expressions, such as `integer()`, `noEvent()`, multi-output function call expression, etc, which may cause type and variability incompatible problem, that is variability of equation is not compatible with type of it. In the paper, reason for this problem is discussed, and the way for debugging is introduced.

Section 2 shows basic rules of equation

type calculation. Section 3 analyzes type and variability incompatible problem and concludes corresponding rules for variability and type calculation. Section 4 complements an additional rule for variability calculation for symbolic transformation. Section 5 is the conclusion of this paper.

2. Basic Rules for Equation Type Calculation

There are four basic types of variable of Modelica model, as Real, Integer, Boolean, and String, predefined by Modelica. Correspondingly, there are four basic types for equation. Record equations should be split into basic types. The following rules of equation type calculation can be summarized from Modelica Language Specification (Modelica 3.2, 28, 61, 64) [4]:

Rule 1. The resulting type of equation is the same as of the type compatible expression of two sides.

Rule 2. The resulting variability of equation is the higher variability of two sides.

3. Compatibility of Type and Variability

As we known, Real type variable can hold all kinds of variability, while variability of Integer, Boolean or String Type variable is no higher than discrete. So does expressions and equations. That is the rule:

Rule 3. The resulting type and variability of equation variability must be compatible.

In following part, cases for Rule 3 are introduced, to show how it works.

3.1 integer() and noEvent()

There are kinds of pre-defined built-in function in Modelica (Modelica 3.2, 19, 109). Generally, type and variability calculation of call of these functions follows Rule 1 and 2. However, there are two particular functions, integer() and noEvent(), are quite different. According definition of them, there is a rule for their variability and type calculation.

Rule 4. Variability of integer() is no higher than discrete, and type of it is integer; variability of noEvent() is continuous, and type of it is the same as that of input argument.

integer() is more like a implicit type conversion function [5], like int() in C++, that means variability of integer(x) is no higher than discrete, even x is continuous, and type of it is integer, obviously. So, integer(x) indicates that x must be an independent variable. Here is an example:

```
function fn1
  input Real x;
  input Integer y;
  output Real z;
  annotation (derivative = derfn1);
algorithm
  z:=x^2+y^2;
end fn1;
```

```
function derfn1
  input Real x;
  input Integer y;
  input Real xder;
  output Real zder;
algorithm
  zder:=2*x*xder+y^2;
end derfn1;
```

```
Real x;
Real y;
Real u;
Real v;
equation
```

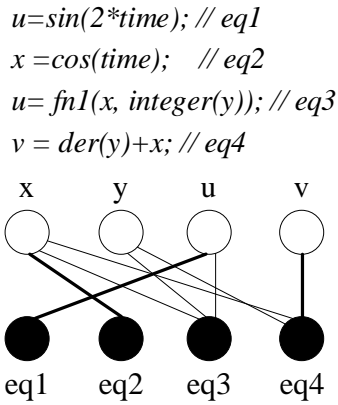


Figure 1 Bipartite graph of equations, with matching edges marked by thick lines

In the example, each equation is legal, all their types are continuous, and here is the bipartite graph for equations of model (Fig. 1). When finding matching in bipartite graph, it should be careful that argument of integer() cannot be the matching vertex with the same equation integer() present in. So, in this case, result of structural analysis is that, y is under-determined, while x and u are over-determined with equation eq1, eq2 and eq3.

noEvent() doesn't trig event as defined (Modelica 3.2, 26), even input argument is an event expression. When we combined noEvent() and integer() together, there was an interesting result. Here is the simple example.

```

Real x;
Integer i = 1;
Integer i2;
equation
    i = x; // eq1, legal
    i2 = noEvent(integer(x)); // eq2,
illegal

```

In this case, eq1 is legal, as variability of eq1 is continuous (higher one of i and x), and type of it is Real (as the same type of type compatible variable of i and x). However, eq2 is illegal, variability of eq2 is continuous (right hand is continuous), while type of eq2 is Integer (type of two hands of eq2 are Integer), that is breach of Rule 3.

3.2 Call Expression of User-define Function

Type and variability calculation of function call follows Rule 5:

Rule 5. Assume that function is defined as single output, variability of function call expression is the same as the higher variability of input real arguments; type of function call is the same as the one of output formal parameter.

Variability of function call expression is the same as the higher variability of all input real arguments, is because that “all assignment statements within function are implicitly treated with the noEvent function” (Modelica 3.2, 87), which means if input arguments are continuous, it is impossible to generate a discrete output. For example:

```

parameter Real a = 10;
parameter Real b = a+2;
parameter Real b = f(a+b, a)*2;

```

Since both a+b and a are parameter, f(a+b, a) is parameter. But if model is:

```

Real a = 10;
parameter Real b = 2;
parameter Real b = f(a,2)*2; //illegal

```

Variability of f(a,2) is the higher variability of a and 2, that is continuous.

A more complex example is like follows:

```

function fn1
    input Real x;
    input Integer y;
    output Real z;
    annotation (derivative = derfn1);
algorithm
    z:=x^2+y^2;
end fn1;

```

```

function derfn1
    input Real x;
    input Integer y;
    input Real xder;
    output Real zder;

```



```

algorithm
  zder:=2*x*xder+y^2;
end derfn1;

function fn2
  input Real a;
  output Integer b;
algorithm
  b:= integer(a);
end fn2;

Real x;
Real y;
Real u;
Real v;
equation
  u=sin(2*time);
  x =cos(time);
  u= fn1(x, fn2(y)); // eq1, illegal
  v = der(y)+x;

```

In eq1, the variability of fn1(x, fn2(y)) is continuous, as x is continuous, and the type of it is Real, then the right hand and left hand have the compatible variability and type. However, the variability of fn2(y) is continuous, and the type of it is Integer, that is breach of Rule 3. We could make a transformation to show this obscure error more clearly. Let introduce Integer variable, like:

```
Integer i = fn2(y); // eq2
```

Then eq1 is conceptually equivalent with:

```
u= fn1(x,i); // eq1', legal
```

After transformation, eq1' is legal, but eq2 is a wrong equation, obviously, as type of variable i is discrete, and variability of fn2(y) is continuous. It is impossible to assign a continuous value to a discrete variable.

3.3 Symbol “.”

“.” is a symbol for member access. Let extend its meanings to present split pattern of multi-output function call.

For call expression of multi-output function, variability and type calculation follows Rule 5, with a split transformation of function call. That is, equation that contains multi-output function call expression should be split into basic types before type calculation. Take following case as an example:

```

function fn
  input Real x;
  input Real y;
  output Real u;
  output Integer v;
algorithm
  u := x+y;
  v := integer(x-y);
end fn;

Real a,b,c,d;
Integer k;
equation
  c=3*sin(time);
  d=cos(time);
  (a,k)=fn(c,d); // eq1
  b=fn(2,if c>0 then 3 else -0.5); //eq2

```

In this case, eq1 should be split into following equations:

```
a=fn(c,d).u; // eq1-1
```

```
k=fn(c,d).v; // eq1-2
```

And eq2, though it is a basic type equation, should be equivalently transformed into:

```
b=fn(2,if c>0 then 3 else -0.5).u
```

Key of split transformation is to put the corresponding output formal parameter at the right position, with a symbol “.”, as a member attached to its parent expression.

With these transformations, type of multi-output function call expression could be calculated by Rule 5. For example, the type of right hand of equation $k=fn(c,d).v$ is the type of v, that is Integer, and the type of equation is Integer. Variability of right hand is continuous (higher variability of c and d), and variability of equation is continuous. It

indicates that there is an error in equation $(a,k)=fn(c,d)$, with a breach of Rule 3.

3.4 If-Expression

If-expression is defined as “if expression1 then expression2 else expression3” (Modelica 3.2, 19). Rule for variability and type calculation of if expression is:

Rule 6. Variability of if-expression is the highest variability of expression1, expression2 and expression3, type of it is the type of expression2.

Type of if-expression is type of expression2, as expression2 and expression3 should be defined as type compatible, while expression1 affects variability of if-expression. For example:

```
Integer x = if noEvent(time > 0) then 1 else 2;
```

Variability of equation in the example is continuous, as that of right hand is continuous, following Rule 6, while type of equation is Integer, as both hands are Integer. Thus, resulting variability and type breach Rule 3, means that equation is illegal.

3.5 Event Expressions

For event expressions, like event triggering mathematical functions (Modelica 3.2, 21), relational expressions, etc, calculation rule is:

Rule 7. Variability of event expression is no higher than discrete, unless it is present in when-clause.

For example:

```
Integer x;
equation
  when time > 0 then
    x = if noEvent(time > 0) then 1 else 2;
  end when;
```

For equation $x = \text{if noEvent}(time > 0)$ then 1 else 2 is present in when-clause,

variability of equation is discrete, different with example in section 3.4, and type of it is Integer. So, in this case, Rule 3 is followed, and the equation is legal.

4. Rule for Symbolic Transformation

In the case where function call is inlined, part of assignment statements will become the part of equations, and the inlined result should be treated with noEvent function.

Here is an example:

```
function f
  input Real in1;
  input Real in2;
  output Integer out1;
  annotation(Inline=true);
  algorithm
    out1 := if in1 > 0 then in1 else in2;
  end f;
  Real x;
  Integer i = 2;
equation
  i = f(x, time); // eq3
```

When $f(x, \text{time})$ of eq3 is inlined, the inlined result should be:

```
i = if noEvent(x > 0) then x else time;
```

rather than:

```
i = if x > 0 then x else time;
```

It is concluded as:

Rule 8. Symbolic transformation must not change basic type and variability of equation.

5. Conclusion

Type calculation of Modelica equation has been researched and a serial of rules for variability and type calculation has been concluded in the paper. Kinds of expression

are analyzed to explain possible variability and type incompatible problem, and more rules are introduced, with several examples to show how rules work. The rules for variability and type calculation for equation will help to find out obscure errors in the model(such as examples in section 3), and to build more accurate bipartite graph for DAEs from Modelica model.

REFERENCES

- [1]. Peter Bunus, Peter Fritzson. Methods for Structural Analysis and Debugging of Modelica Models. Proceedings of the 2nd International Modelica Conference, 2002, 10: 157~165
- [2]. Ding Jianwan. Research on Methods for Consistency Analysis and Reduction of Declarative Simulation Models: [PhD thesis]. China: Huazhong University of Science & Technology, 2006
- [3]. David Broman. Types in the Modelica Language. Proceedings of the 5th International Modelica Conference, 2006, 9:303~315
- [4]. Modelica Language Specification V3.2. <https://www.modelica.org/>.
- [5]. Peter Fritzson. Principles of Object-Oriented Modeling and Simulation with Modelica 2.1. Wiley-IEEE Press, New York, USA, 2004
- [6]. Futong Lv. Numerical Computing Methods, chapter 5. Tsinghua University Press, 2008.

Shock Absorber Modeling and Simulation Based on Modelica

Yuming Hou, Lingyang Li, Ping He, Yunqing Zhang, Liping Chen
CAD Center, Huazhong University of Science and Technology, China
zhangyq@hust.edu.cn

Abstract

The purpose of shock absorbers are to dissipate impact energy, and control tire force variation, the shock absorber has great influence on both ride and handling performance of vehicles, and a great many previous researches have been done on modeling and simulation of the shock absorber. In this paper, a detailed model of shock absorber is established, which contains rebound chamber, compression chamber, piston valve assembly, base valve assembly and so on. Those models are built using modelica language, modelica is a language for modeling of physical systems, designed to support effective library development and model exchange. It is a modern language built on a causal modeling with mathematical equations and object-oriented constructs to facilitate reuse of modeling knowledge.

Keywords: shock absorber; ride; handling; Modelica

1 Introduction

Shock absorber is widely used on vehicle. The purposes of the shock absorber are to dissipate the energy accumulated by the suspension spring displacement. The damping of the shock absorber for compression motion is usually less than that of rebound motion, in such a case, less force is transmitted to the vehicle when crossing a bump. By comparison, the shock absorber provides more damping force for rebound motion in order to dissipate energy stored in the suspension system quickly [1]. When the shock absorber is operated, hydraulic oil is passed between chambers via a system of hydraulic valves, and the damping effect is accomplished by the resistance of the oil when flowing through the valves. For the importance of the shock absorber on vehicle ride and handling performance, it is necessary to establish an accurate mathematical model of shock absorber, and there is a great wealth of literatures devoted to the modeling and simulation of the shock absorber. Herr [2] presented a computational fluid dynamics method

combined with a dynamic modeling technique. Which used to study the flow and performance of automotive hydraulic dampers / shock absorbers. Simms [3] established a non-linear hysteretic physical shock absorber model, and the processes utilized to identify the constituent parameters, and the model is validated by comparing simulated results to experimental data for a test damper, for three discrete frequencies of sinusoidal excitation of 1, 3 and 12 Hz. Talbot [4] presented a mathematical model of a gas-charged mono-tube racing damper. The model includes bleed orifice, piston leakage, and shim stack flows, and also includes models of the floating piston and the stiffness characteristics of the shim stacks. Chavan [5] study the damper lag and hysteresis which are the important parameters affecting the dynamic response of the hydraulic shock absorbers, and the response of the suspension unit to road excitation strongly influences motorcycle ride comfort.

Modelica is a freely available, object-oriented language for modeling of large, complex, and heterogeneous physical systems. It is suited for multi-domain modeling, for example, mechatronic models in robotics, automotive and aerospace applications involving mechanical, electrical, hydraulic and control subsystems, process oriented applications and generation and distribution of electric power. Models in Modelica are mathematically described by differential, algebraic and discrete equations. No particular variable needs to be solved for manually. A Modelica tool will have enough information to decide that automatically. Modelica is designed such that available, specialized algorithms can be utilized to enable efficient handling of large models having more than one hundred thousand equations. Modelica is suited and used for hardware-in-the-loop simulations and for embedded control systems.

2 Shock Absorber Modeling

The shock absorber is one of the most important elements in a vehicle suspension system. For analyzing the influence of the uncertain parameters on the response of the shock absorber, a detailed analytical

model of the shock absorber is necessary. The first step of establishing the analytical model is to understand the physics of the shock absorber. Fig. 1 shows the schematic of the shock absorber. The shock absorber mainly consists of rebound chamber, compression chamber, reserve chamber, piston valve assembly and base valve assembly. When the piston valve assembly moves up and down, pressure differential is generated between the rebound chamber and the compression chamber, also there is pressure differential between the compression chamber and the reserve chamber. The pressure differential forces the fluid to flow through the valves and orifices. The damping force during the compression and rebound stroke are produced on account of the resistance offered by the fluid in flowing through the valves and orifices [6].

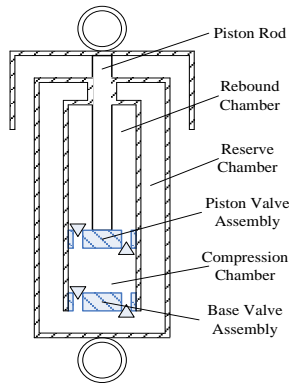


Fig. 1 The Schematic of the shock absorber

During the compression, the piston rod moves down and pushes the fluid in the compression chamber flowing to the rebound chamber and reserve chamber. The detailed structures of the piston valve assembly and the base valve assembly are given in Fig. 2, the fluid paths during the compression are also shown in the figure.

As shown in Fig. 2(a), the by-pass valve and rebound valve are essential non-return valves, during the compression, the rebound valve disc is closed, but there are four orifices on the rebound valve disc allow the fluid flow through, cross the holes of piston body and by-pass valve disc to the rebound chamber. The by-pass valve disc opens only if the pressure differential between the compression chamber and the rebound chamber reaches the preload of the by-pass valve spring. In Fig. 2(b), the check valve is closed, and before the pressure differential between the compression chamber and reserve chamber reaches the preload of the compression valve disc, the compression valve will be closed. But the fluid can still pass through the four orifices on the compression valve disc. When the pressure differential exceeds the preload, the compression valve will be opened.

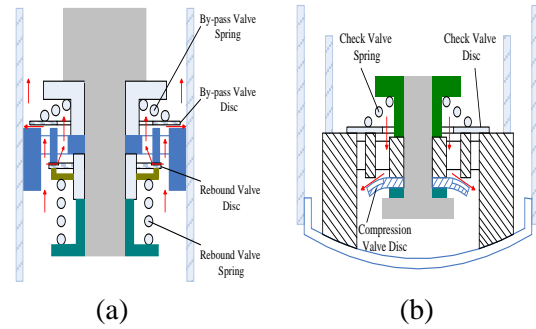


Fig. 2 Fluid paths through the piston valve and base valve during the compression stroke

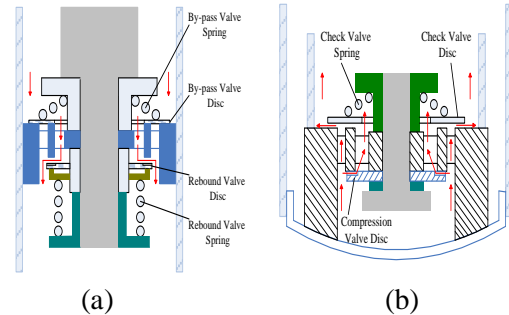


Fig. 3 Fluid path through the piston valve and base valve during the rebound stroke

During the rebound, fluid in the rebound chamber was forced flowing to the compression chamber, the missing oil equivalent to the rod volume extracted from the rebound chamber is complemented from the reserve chamber to the compression chamber, and the flow paths are shown in Fig. 3.

In Fig. 3(a), the by-pass valve is closed, when the pressure differential between the rebound chamber and the compression chamber is lower than the preload of the rebound valve spring, the rebound valve is also closed, but the fluid will pass through the orifices on the disc. And when the pressure differential reaches the preload, the rebound valve disc will be opened. As depicted in Fig. 3(b), the compression valve is closed, and the fluid will flow through the orifices on the disc from reserve chamber to compression chamber. And the check valve will be opened when the pressure differential between the reserve chamber and the compression chamber reaches the preload of the check valve spring.

From discussed above, an equivalent scheme of the shock absorber is built in Fig. 4.

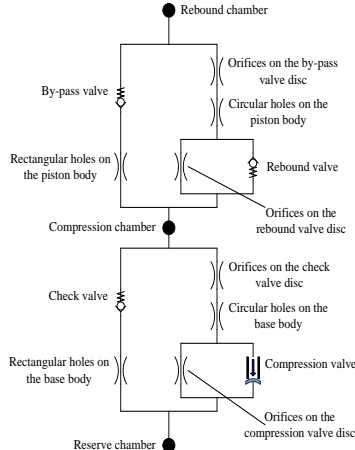


Fig. 4 The equivalent scheme of the shock absorber

For a generic orifice or a hole, the flow rate Q can be expressed as a function of the pressure drop Δp :

$$Q = CA \sqrt{\frac{2\Delta p}{\rho}} \quad (1)$$

Where, C is the flow coefficient, A denotes the flow area of the orifice, ρ is the density of the oil.

The modelica code of the orifice is given in Fig. 5.

```

model Constant
  Damper.Interfaces.HyLibProps Prop;
  import Modelica.Constants.pi;
  parameter Real Cq = 0.7 "The flow coefficient";
  parameter Real area = 1.5e-006 "The cross area";
  parameter Real rho = 900 "The density of the oil";
  parameter Real k1(unit = "1", min = 0) = 10. "laminar part of orifice model";
  final parameter Real k2 = 1 / (Cq * Cq);
  final parameter Real hd = 2 * sqrt(area / pi);
  Real dp;
  Real q;
  Interfaces.Hydraulic.Port_A Port_A_Constant annotation (extent = [-110, -10; -90, 10]);
  Interfaces.Hydraulic.Port_B Port_B_Constant annotation (extent = [90, -10; 110, 10]);
  annotation (Diagram, Icon);
equation
  dp = Port_A_Constant.p - Port_B_Constant.p;
  q = (sqrt(k1 ^ 2 * Prop.nu ^ 2 * Prop.rho ^ 2 + 8 * hd ^ 2 * k2 * noEvent(abs(dp)) * Prop.rho)
    - k1 * Prop.nu * Prop.rho) * hd * Modelica.Constants.pi / (8 * k2 * Prop.rho);
  Port_A_Constant.q = noEvent(if dp >= 0 then q else -q);
  Port_B_Constant.q + Port_A_Constant.q = 0;
end Constant;
    
```

Fig. 5 The modelica code of the orifice

For a check valve with spring preload, the formula given below describes the relation between the flow rate Q_v and pressure drop Δp_v :

$$Q_v = \begin{cases} 0 & \text{if } \Delta p_v \leq \Delta p_0 \\ C l_b \frac{(\Delta p_v - \Delta p_0) A_{disc}}{k_s} \sqrt{\frac{2\Delta p_v}{\rho}} & \text{if } \Delta p_v > \Delta p_0 \end{cases} \quad (2)$$

Where, Δp_0 is the preload of the spring, k_s denotes the stiffness of the spring, l_b represents hydraulic perimeter of the valve disc. A_{disc} is the area which oil pressure acts on the valve disc. The modelica code of the check valve is shown in Fig. 6.

```

model CheckValve
  extends Valves.CheckValvePartial;
  annotation (Documentation(info = "<HTML><P>Model</P><CheckValveNoStates
  parameter Modelica.SIunits.Pressure pClosed(final min = 0) = 100000. "pressure to start opening the valve";
  parameter Modelica.SIunits.Pressure pOpen(final min = 0) = 125000. "pressure to open valve completely";
  parameter Modelica.SIunits.Diameter diameter(min = 0.1) = 1.e-003 "diameter of equivalent orifice";
  parameter Real Gleak = 1.e-012 "conductance of leakage";
  parameter Real k1(unit = "1", min = 0) = 10. "laminar part of orifice model";
  parameter Real k2(unit = "1", min = 0) = 2. "turbulent part of orifice model, k2 = 1 / C_d^2";
  Boolean closed(start = false) "valve closed, only leakage";
  Boolean open(start = false) "valve wide open";
  Real pOpen;
  Modelica.SIunits.VolumetricFlowRate qOpen;
  annotation (Icon);
equation
  closed = dp < pClosed;
  open = dp > pOpen;
  qOpen = (sqrt(k1 ^ 2 * Prop.nu ^ 2 * Prop.rho ^ 2 + 8 * diameter ^ 2 * k2 * noEvent(abs(pOpen)) * Prop.rho)
    - k1 * Prop.nu * Prop.rho) * diameter * Modelica.Constants.pi / (8 * k2 * Prop.rho);
  pOpen = qOpen / (pOpen - pClosed);
  Port_A.q = if closed then dp * Gleak else if open then (sqrt(k1 ^ 2 * Prop.nu ^ 2 * Prop.rho ^ 2
    + 8 * diameter ^ 2 * k2 * noEvent(abs(dp)) * Prop.rho)
    - k1 * Prop.nu * Prop.rho) * diameter * Modelica.Constants.pi / (8 * k2 * Prop.rho)
    + dp * Gleak else (dp - pClosed) ^ 2 * pOpen / (pOpen - pClosed) + dp * Gleak;
  assert(pOpen > pClosed, "Parameter pOpen MUST be greater than parameter pClosed.");
end CheckValve;
    
```

Fig. 6 The modelica code of the check valve

The mathematical model of the compression valve is presented as below:

$$Q_c = \begin{cases} 0 & \text{if } \Delta p_c \leq \Delta p_{c0} \\ C l_{bc} \delta \sqrt{\frac{2\Delta p_c}{\rho}} & \text{if } \Delta p_c > \Delta p_{c0} \end{cases} \quad (3)$$

$$\delta = \frac{\Delta p_c}{E_c h^3} [(9r_1^4 + 8r_1^3 r_2 - 18r_1^2 r_2^2 + r_2^4) / 6 - 4r_1^3 r_2 \ln(r_1 / r_2)] \quad (4)$$

Where, Q_c is the flow rate through the compression valve, Δp_c is the pressure drop, l_{bc} denotes the hydraulic perimeter of the compression valve disc, δ represents the deflection of the disc, E_c is the elastic modulus of the disc, h is the thickness of the disc. r_1 and r_2 are the outer and inner contact radius between the disc and the compression valve body. Fig. 7 shows the modelica code of the compression valve.

```

model CompressionValve
  Damper.Interfaces.HyLibProps Prop;
  import Modelica.Constants.pi;
  parameter Modelica.SIunits.Force Fpre = 10 "The preload of the valve plate";
  parameter Modelica.SIunits.Area A = 2.e-004 "The area of the valve plate";
  parameter Modelica.SIunits.Length lp = 8.e-002 "The perimeter of the valve plate";
  parameter Modelica.SIunits.Length h = 4.5e-004 "The thickness of the valve plate";
  parameter Modelica.SIunits.Length r1 = 1.9e-002 "The outer radius of the valve plate";
  parameter Modelica.SIunits.Length r2 = 8.e-003 "The outer radius of the washer";
  parameter Real E = 200000000000. "The elastic modulus";
  parameter Real Cq = 0.7 "The flow coefficient";
  parameter Real Aleak = 1.e-007 "The leak area";
  parameter Real rho = 900 "The density of the oil";
  parameter Real k1(unit = "1", min = 0) = 10. "laminar part of orifice model";
  final parameter Real pcrack = Fpre / A;
  final parameter Real k2 = 1 / (Cq * Cq);
  Real dp;
  Real q;
  Real delta "The bend deflection of the valve plate";
  Real area;
  Real hd;
  Interfaces.Hydraulic.Port_A Port_A_Constant annotation (extent = [-110, -10; -90, 10]);
  Interfaces.Hydraulic.Port_B Port_B_Constant annotation (extent = [90, -10; 110, 10]);
  annotation (Diagram, Icon);
equation
  dp = Port_A_Constant.p - Port_B_Constant.p;
  if noEvent(dp <= pcrack) then
    delta = 0;
  else
    delta = (dp - pcrack) * ((9 * r1 ^ 4 + 8 * r1 ^ 3 * r2 - 18 * r1 ^ 2 * r2 ^ 2
      + r2 ^ 4) / 6 - 4 * r1 ^ 3 * r2 * ln(r1 / r2)) / (E * h ^ 3);
  end if;
  area = delta * lp + Aleak;
  hd = 2 * sqrt(area / pi);
  q = (sqrt(k1 ^ 2 * Prop.nu ^ 2 * Prop.rho ^ 2 + 8 * hd ^ 2 * k2 * noEvent(abs(dp)) * Prop.rho)
    - k1 * Prop.nu * Prop.rho) * hd * Modelica.Constants.pi / (8 * k2 * Prop.rho);
  Port_A_Constant.q = noEvent(if dp >= 0 then q else -q);
  Port_B_Constant.q + Port_A_Constant.q = 0;
end CompressionValve;
    
```

Fig. 7 The modelica code of the compression valve

3 Simulation

A shock absorber simulation model is established using MWorks/Modelica software, as shown in Fig. 8, the model contains the rebound valve, by-pass

valve, check valve, compression valve, rebound chamber, compression chamber, reserve chamber, the mass of the piston and so on. The input of the simulation model is a sine displacement to the piston rod. And Fig. 9 shows the response of the damping force vs the velocity of the piston. Fig. 10 shows that the response of the damping force vs the displacement of the piston

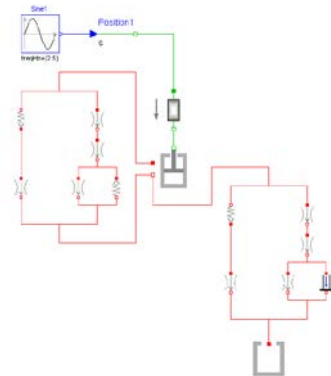


Fig. 8 The simulation model of the shock absorber

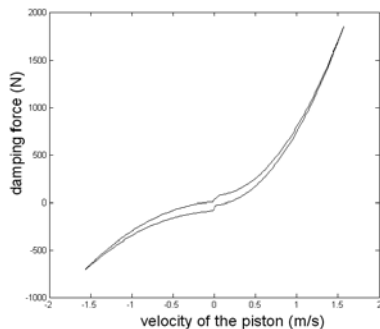


Fig. 9 Response of the damping force vs the velocity of the piston

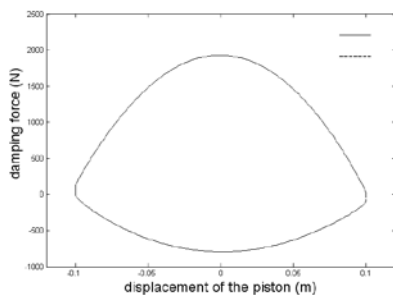


Fig. 10 Response of the damping force vs the displacement of the piston

4 Conclusions

In this paper, the mathematical model of the shock absorber which contains rebound chamber, compression chamber, piston valve assembly, base valve assembly is given in the paper, then a simulation model is established using MWorks/modelica software, the simulations are performed to evaluate the damping force of the shock absorber. The results shows that

the Modelica language is available for modeling multi-domain physical system.

Acknowledgement

This work was supported by the National High-Tech R&D Program, China (No. 2009AA044501).

References

- [1] S. Subramanian, R. Surampudi and K. R. Thomson. "Development of a Nonlinear Shock Absorber Model for Low-Frequency NVH Applications," SAE paper 2003-01-0860.
- [2] F. Herr, T. Mallin, J. Lane and S. Roth. "A Shock Absorber Model Using CFD Analysis and Easy5," SAE paper 1999-01-1322.
- [3] A. Simms and D. Crolla. "The Influence of Damper Properties on Vehicle Dynamic Behaviour," SAE paper 2002-01-0319.
- [4] M. S. Talbott and J. Starkey. "An Experimentally Validated Physical Model of a High-Performance Mono-Tube Damper," SAE paper 2002-01-3337.
- [5] C. B. Chavan, M. K. Venkata, R. Babu and R. Bharat. "Experimental Study on Effect of Damper Lag on Motorcycle Ride Comfort," SAE paper 2006-32-0096.
- [6] L. C. Andreotti and S. N. Vannucci. "Shock Absorber Mathematical Modeling," SAE paper 982959.

Integrating occupant behaviour in the simulation of coupled electric and thermal systems in buildings

Ruben Baetens* Dirk Saelens

Building Physics Section, Department of Civil Engineering, K.U.Leuven, Belgium

* *ruben.baetens@bwk.kuleuven.be*

Abstract

The presented work depicting the integrated modelling of probabilistic occupant behaviour in buildings consists of non-physical modelling with a physical multi-domain impact. The human behaviour considering occupancy, the use of lighting and the use of electric appliances in dwellings has been implemented, but the same method can be used for other stochastic behaviour. The stochastic behaviour is used for simulation of coupled thermal and electrical systems in the building stock and is of high importance for the assessment of smart grids and distributed energy generation. Implementing stochastic occupant behaviour influences the internal heat gains which in turn influence the heat load of the building and the switch-on and -off moment of e.g. an electric heat pump. This, together with the power demand of the used electric appliances and possible on-site generation determine the load on the electric grid and possible instabilities. Here, the use of deterministic profiles for use at both the building and the building district scale no longer fits.

Comparison between a deterministic approach as proposed in ISO 13790 and the use stochastic profiles shows that the direct *first order* effect is *on average* rather small: the difference in total internal gains and its influence on the indoor temperature averages nearly zero and the standard deviations σ are small, however high peaks may occur. Also the difference in effect on the electric distribution grid voltage averages nearly zero, however here strong peaks occur which are of most importance for the grid stability. When taking in account the *second order* effect of heating by means of electricity, much larger differences are noticed: due to longer and more differentiated occupancy times, the average indoor temperature rises. Furthermore, the moment of heating differentiates compared to a deterministic approach resulting in more but smaller peak demands towards the electricity grid.

Keywords: Stochastic modelling; Occupant be-

haviour; Grid load; Thermal building response.

1 Introduction

Since the development of dynamic building simulation programs such as TRNSYS and ESP-r in the mid-70's [14, 26], the assessment of comfort and energy demand of buildings has been subject of intensive research. As a recast of the European legislation 2002/91/EG obliges all members to build *nearly* zero-energy buildings by 2020, the need of detailed dynamic simulations still increases. The recast should result in the implementation of renewable energy in the building stock, most often resulting in an all-electrical solution with a combination of building-integrated photovoltaic systems (BIPVs) and an electrical heat pump where the total electricity consumption is. However, a problem of simultaneity between electricity production and consumption arises with the distribution grid as virtual storage [3, 4]. This paper focuses on the modelling of user behaviour, influencing both the fluctuations in the electricity demand as well as the thermal demand.

2 Stochastic behaviour

The behaviour of building occupants is most often simplified in current practice as deterministic schedules of user behavior as inputs to the building simulation model (e.g. as in ISO 13790 [12]), whereas the probabilistic user behaviour plays an important role. The combination of both stochastic and controllable local service demand, and both stochastic and controllable local energy conversion in different energy vectors (i.e. heat, cold, fuel or electricity in buildings) allows to operate [7] and optimize the energy distribution and control in many different ways at both the scale of a single building and the building district.

The modelling of human behaviour is implemented in Modelica as it is part of a bigger approach including

thermal building simulation, simulation of thermal energy systems, electrical energy systems and distributed generation (see Fig.1).

The presented work consists of the non-physical modelling of probabilistic occupant behaviour in buildings with a physical multi-domain impact on both thermal and electrical aspects. The behaviour considering occupancy [20], the use of lighting [22] and the use of use of appliances [17] in dwellings has been implemented, but the same method can be used for other stochastic behaviour (e.g. opening of windows by occupants). The implemented stochastic model is largely consistent with the model of Richardson et al. [21]. The output of the model are presence and activity profiles of the building occupants, coupled to the usage of electric appliances and lighting which result in (i) convective Q_c and radiative Q_r internal heat gains and (ii) the real or active electric power demand P . The model is integrated in the dynamic simulation of the thermal response of buildings and its grid impact towards the assessment of distributed generation at district scale.

Many researchers [10, 18, 19, 20, 21, 22, 23, 25, 30] use Markov properties for modelling occupancy and the use of appliances in buildings. A Markov process is a stochastic process $\{X_t, t \geq 0\}$ with values in a state space E where for any $s < t$ and any measurable set $A \subset E$ holds that $P(X_t \in A | X_r, 0 \leq r \leq s) = P(X_t \in A | X_s) = P(s, X_s, t, A)$ where the function $P(s, X_s, t, A)$ describes the probability that the process is in A at t conditioned by the information that the process is in x at time s [15]. The Markov property depicts that the future state does not depend on how the current state is obtained but only depends on the present state itself. The process is characterized by the *transition probabilities* $A \rightarrow P(s, X_s, t, A)$ parameterized by s, x and t .

Within this work, both *embedded discrete time Markov chains* and *semi-Markov processes* are implemented.

2.1 Occupancy: Embedded discrete time Markov chains

Occupancy in buildings is typically dealt with as embedded discrete time Markov chains [18, 20, 23, 30] however also semi-Markov processes may be found suitable [19], i.e. especially for single-person offices where only two states are possible. Although used only for evaluation of uccpancy, the implemented model for embedded discrete time Markov chains is generic and can be used and extended for all similar Markov chains.

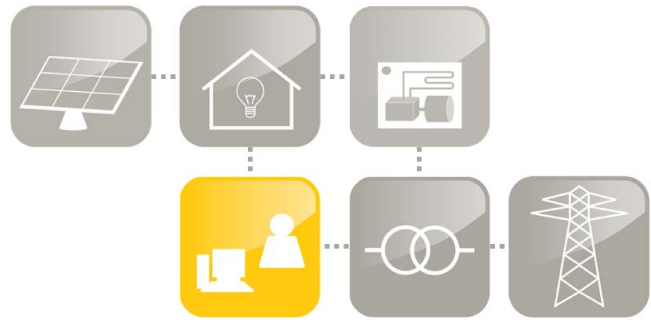


Figure 1: *Occupant behaviour as part of the simulation of energy networks at district scale by the K.U.Leuven Energy Institute.*

In an embedded discrete time Markov Chain, the possible change in occupancy (i.e. a person entering or leaving the building) is evaluated repeatedly after a discrete time step (i.e. mostly 10 minutes) based on the previously described Markov property $P(s, X_s, t, A)$. The Markov-properties are stored in records as

```
record Occupance(Integer n(min=1),Sl.Time period,
  Sl.Time s, Real[s,n+1,n+1] Twd, Twe);
```

where n depicts the number of different states, *period* the total time span across which the Markov process transition probabilities repeat themselves (i.e. mostly 24 hours) and s the number of equal time steps within the period wherefor the transition probabilities are given.

The matrices T_{wd} and T_{we} depict the transition probabilities of the Markov process for a work day and weekend respectively so that $P(s, X_s, t, A) = T[s, X_s + 1, A + 1]$. T_{wd} and T_{we} are retrieved from Richardson et al. [20] where they are given for dwellings with 1 to 5 inhabitants and where the possibility of correlation between different inhabitants for arrival or departure is included. At each discrete time step, the the actual current (cumulated) transition probabilities based on the knowledge of the present state and the time of day are extracted from the \mathbf{R}^3 -matrix containing all transition probabilities and evaluated for possible state changes.

```
model OccStoch(Occupance (...))
  MSL.IntegerOutput occ;
  MSL.HeatPort Qconv, Qrad;
end OccStoch;
```

The presence of inhabitants determines internal gains from people and the possibility of opening windows, whereas it is a necessary condition for the use of many electric appliances and the use of lighting in buildings.

2.2 Appliances: Semi-Markov processes

Differently from the evaluation of occupancy in buildings, the use of appliances is generally implemented as semi-Markov processes [19, 21, 22, 25, 30].

For the use of many domestic electric appliances, presence of at least one of the occupants is a first necessary condition, derived from the previously mentioned embedded discrete time Markov chains.

Secondly, the activity profile of humans depends on the moment of the day. Activity profiles are generally described as a sequence of probabilities denoting the chance that a certain activity occurs at the time of a day. The probabilities are depicted independently from the current state (i.e. doing the activity or not) but depending on the number of active occupants, differently from the description of human occupancy in buildings by transition probabilities which depend on the current state.

```
record Activity(Integer n(min=1), SI.Time period,
  SI.Time s, Real[s,n+1] Pwd, Pwe);
```

```
model ActProb(Activity[:] (...))
  MSL.IntegerInput occ;
  output Boolean[:] act;
end ActProb;
```

where n depicts the number of occupants active at moment t derived from the stochastic occupancy profile for which the stated probabilities count, $period$ the total time span across which the Markov process probabilities repeat themselves (i.e. mostly 24 hours) and s the number of equal time steps within the period wherefor the transition probabilities are given. The matrices P_{wd} and P_{we} depict the probabilities of the Markov process for a work day and weekend respectively. The different probabilities are derived from Richardson et al. [21] for the activities *watching television*, *cooking*, *doing laundry*, *ironing*, *cleaning the house* and *spending time on washing and clothing*. The library of activities can be extended with additional data from time-consumption surveys [28]. Currently, the implementation only takes into account the different profiles for working days and weekend days, but can be easily expanded e.g. for taking into account long days of absence or vacation.

At each discrete time step, the activity probability is evaluated for possible use of appliances related to the activity based on the relative use of appliances. When the *decision* is taken that a certain appliance is switched on, the length during which the appliance

will remain on is determined once, differently from the occupancy pattern where the possible switch is evaluated for every discrete time step. This simplification of an embedded discrete time Markov chain into a semi-Markov process is only possible if only two states occur, i.e. *off* and *on* in this case, resulting in less *simulation events* and thus shorter simulation times. The drawback of this approach is that interactions from one occurrence to another are excluded, i.e. it excludes adaptation and intermediate states.

The data implementation happens as

```
record Appliance(Activity act, Real ncycle, Real cal,
  Real frad, Real fconv, SI.Time lcycle, SI.Power Pcycle,
  SI.Power Pstandby);
```

```
record Light(Integer n,SI.Power[n] Pcycle);
```

where act is the required activity of the appliance, n_{cycle} is the average number of cycles during a year, cal a calibration scalar defining the relation between the activity and the effective use of the appliance, and l_{cycle} is the average length of usage, where P_{cycle} and $P_{standby}$ are the total power demand of the appliance when switched on and at standby modus respectively and where f_{conv} and f_{rad} are the convective and radiative fraction respectively of the local heat production by the appliances.

For determination of the duration for which an appliance remains on, the distribution of the duration $P(t)$ for the use of lighting is fitted as $0.1664 \ln(t) + 0.1084$ with a r^2 value of 0.9961 [21] and where t is the duration in minutes. The duration probability for the use of other appliances is set equal to the average usage with a standard deviation of 10 percent except for the television where a $P(t)$ of $1 + 1.021 \exp(t^{0.91})$ is given [21].

2.3 Physical impact

As mentioned earlier, human behaviour in buildings has a multi-physical impact. Occupants presence and the use of electric appliances determine the internal heat gains in a building, whereas the electric appliances (together with possible local electric generation, e.g. by means of a photovoltaic system) determine the load of a building on the electric distribution grid affecting the grid voltage.

Both the occupancy of humans in the building and the use of electric appliance result in internal gains in a building, influencing both thermal comfort and the related energy consumption for heating (and cooling).

For both human presence and each different implemented appliance, the internal heat production is depicted as a (long-wave) radiative f_{rad} (–) and convective f_{conv} (–) fraction. So far, dynamic effects such as wasted heat (e.g. for cooking or washing) or temperature-dependent fractions depending on the cycle are not included.

The fractions of f_{rad} and f_{conv} for different appliances and humans are derived from ASHRAE Fundamentals [2]. When no value is available, a division of $f_{rad} = f_{conv} = 0.5$ is set.

On the electric side, all loads of the the used appliances are seen as active loads P (W). Differently from e.g. heat pumps or motors, where also the reactive power S is of importance.

2.4 Random number generation

For the purpose of probabilistic simulation, the 4-cycle generation algorithm for pseudo-random numbers of Wichmann and Hill [29] is implemented combining a long period of 2^{121} with a small size of state of 16 bytes. Taking into account a discrete time step of 1 minute and the number of stochastic processes in a classic dwelling (i.e. in the order of size between 30 and 100 depending on the number of occupants and appliances) regarding building occupant behaviour, the period of the random generator remains in the order of size of 2^{90} making it suitable for the mentioned modelling purpose as no repetition will occur during a simulation.

3 Physical relevance

The stochastic behaviour of building occupants influences both the electric consumption of the used domestic appliances and the heat production of the same domestic appliances and the occupants. The results show both similarities (i.e. similar averages) as large differences (i.e. large deviations) from the deterministic approach as in ISO 13790 [12] where a deterministic scheme for simulation purposes is depicted on the internal gains from occupants and appliances (see Fig.2,3,4). The standard differentiates three different time zones between 7 pm, 17 am and 23 for the internal gains.

In order to quantify the difference between a stochastic and deterministic approach, a single-zone reference building is modelled (see appendix A) and 5 identical models (with each the same deterministic or different stochastic profile respectively) are coupled to

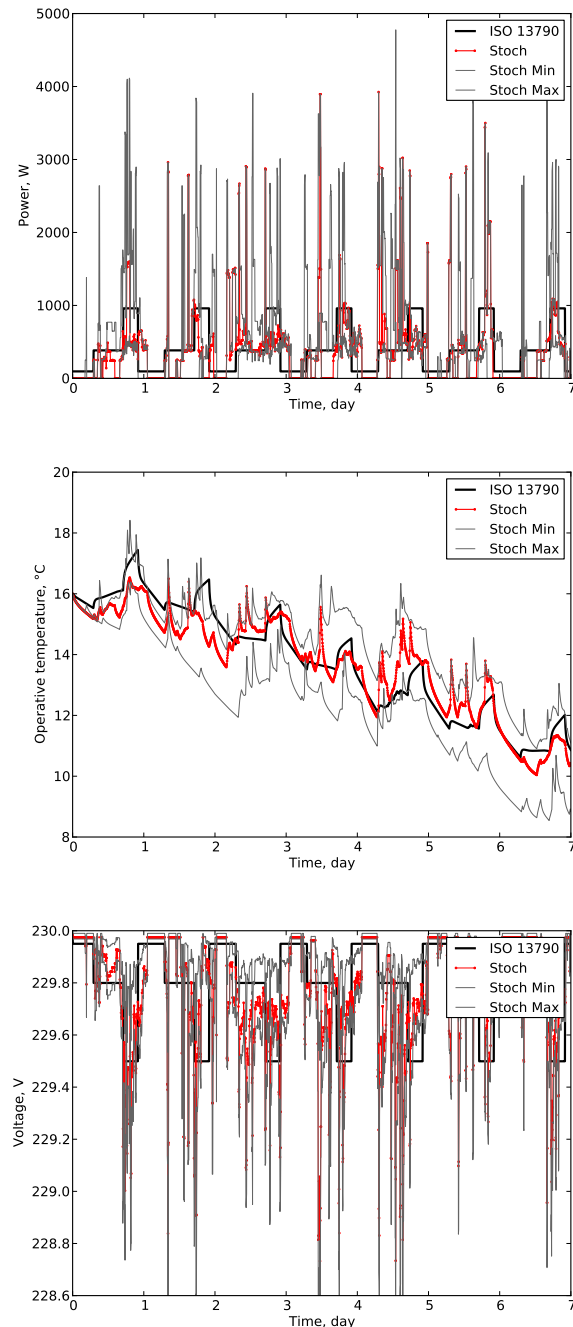


Figure 2: Week profile of (a) the total power, (b) the indoor operative temperatures of a building in free run and (c) the resulting voltage in an example grid as deterministically described in (ISO 13790) and (Stoch) with a stochastic profile of occupancy, lighting and electric appliances. The minima (Stoch min) and maxima (Stoch max) of the results for 1 to 5 inhabitants are given for the same situation with the same appliances.

an electric grid (see appendix B). For quantification of the *second-order* effects of heating, also an ideal heating system has been introduced (see appendix A).

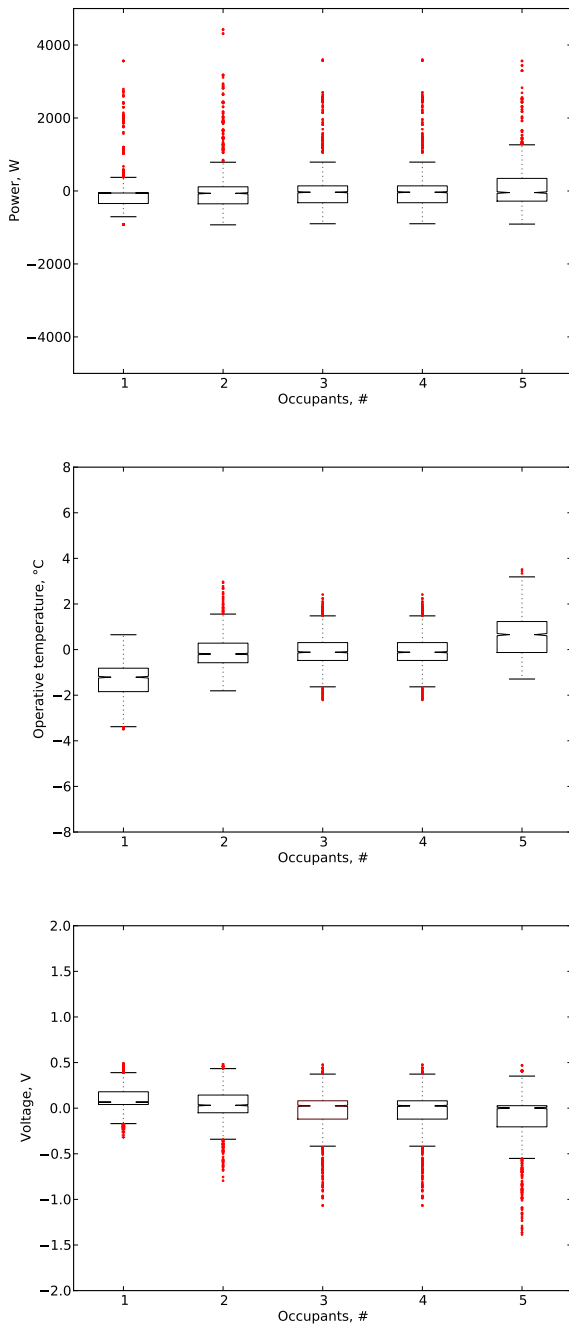


Figure 3: Standard deviations (σ , $-\sigma$) representend by the boxes, (3σ , -3σ) representend by the bars, and tails of difference between the stochastic and deterministic approach for (a) the total electric power demand, (b) the indoor operative temperatures of a building in free run and (c) the resulting voltage in an example grid of the difference between a stochastic profile of occupancy, lighting and electric appliances and as described in ISO 13790 for 1 tot 5 inhabitants.

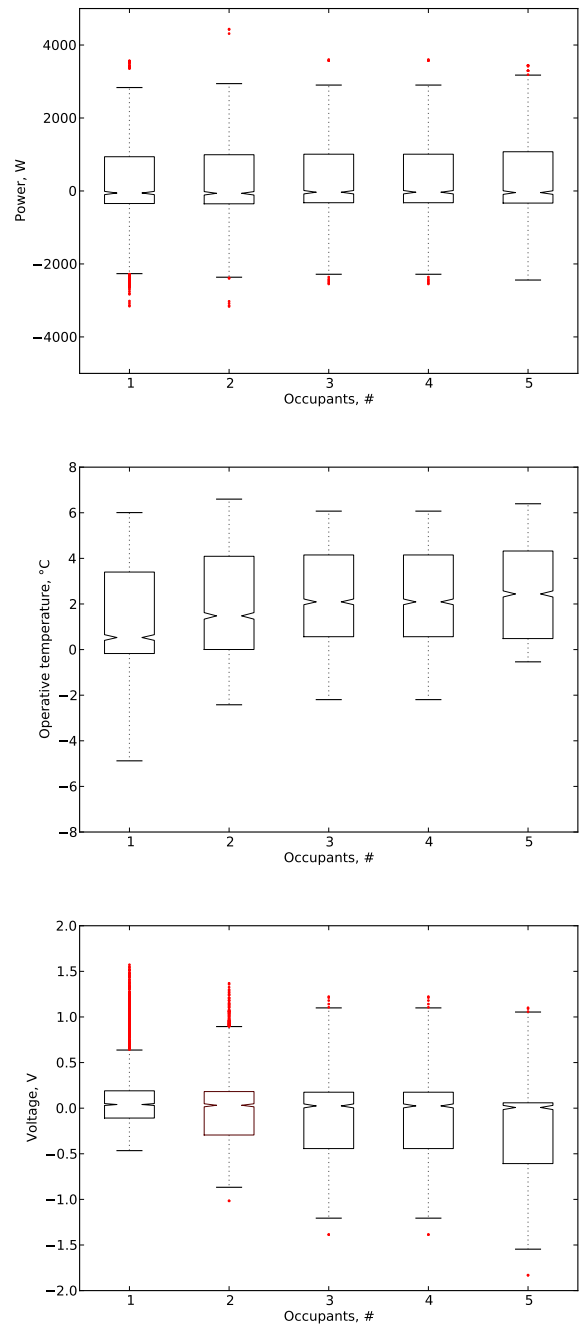


Figure 4: Standard deviations (σ , $-\sigma$) representend by the boxes, (3σ , -3σ) representend by the bars, and tails of difference between the stochastic and deterministic approach for (a) the total electric power demand, (b) the indoor operative temperatures and (c) the resulting voltage in an example grid of the difference between a stochastic profile of occupancy, lighting and electric appliances and as described in ISO 13790 for 1 tot 5 inhabitants and a building including ideal heating.

3.1 Thermal load

The profile of total stochastic and deterministic thermal power (see Fig.2a) results in convective and radia-

tive internal gains for the building and determines the internal temperature (see Fig.2.b), the required thermal energy for thermal comfort and the possibility of overheating.

Comparison between the stochastic and deterministic profile of total power (see Fig.2a,3a) shows both similarities as differences: The difference between the deterministic and stochastic load profiles averages nearly zero determining that the total load is similar, whereas also the standard deviation σ is very small in the order of size of 0.3 kW (see Fig.3a) denoting that also the overall or average day profile, i.e. the moment of the day internal loads occur, is similar in both the deterministic and stochastic profile. In contrast, high peaks e.g. resulting from cooking occur in the stochastic profile which disappear in the deterministic approach (resulting in the long *tail* in Fig.3a).

As the average internal loads are alike between the deterministic and stochastic approach, also the thermal response of an example dwelling (see appendix A) shows small deviations (see Fig.2b,3b) with a standard deviation σ in the order of size of 0.5 Kelvin. As the thermal building response of a dwelling is slow due to a relatively high thermal mass, the effect of the high peaks in the stochastic internal gains remains small compared to a deterministic approach.

3.2 Electric load

The profile of total stochastic and deterministic electric power results in electric currents in the distribution grid and determines the voltage drops (see Fig.2c).

Similar to the thermal load, as the average electric loads are alike between the deterministic and stochastic approach (see Fig.2a), also the average electric response of an example grid (see appendix B): the voltage drop shows small deviations (see Fig.3c) with a standard deviation σ in the order of size of 0.2 Volt.

In contrast to the conclusion on peak loads on the thermal side, electric peak loads have a direct influence on the voltage of an electric grid which disappears in the deterministic approach (see Fig.2c,3c). Even more, the effect of peak loads accumulates at grid-level: as (i) in real life and in the stochastic approach of occupant behaviour not all peaks occur at the same time and (ii) a peak load not only influences the observed voltage at the respective dwelling but also the the observed voltage at the adjacent dwellings between the respective dwelling and the grid source, more and higher voltage drops are noticed in the stochastic approach shown by the long tail of 0.5 to 1.0 Volt in Fig.3c.

As the voltage in a low-voltage distribution grid is

not allowed to drop below 207 V and the strongest voltage drops are caused by peak loads, the use of deterministic profiles may strongly underestimate the possible problem of grid-instability.

3.3 Coupled thermal-electric load

Both the influence on indoor temperature and voltage-drops by the stochastic modelled internal gains are *first-order* effects as they are directly influenced by the bottom-up modelled power profile. As yet described in the introduction, new and future a (zero-energy) buildings results more often in an all-electrical solution with a combination of building-integrated photovoltaic systems (BIPVs) and an electrical heat pump, a trend yet noticeable in existing low-energy dwellings. As the stochastic profile of both occupancy and internal heat gains determine both the switch-on and -off conditions for the electric heating - and thus also the electric peak loads of the heat pump towards the grid - also a *second-order* effect can be noticed. In order to visualize this effect, an *ideal* heating system is implemented in the example dwelling (see appendix A).

3.3.1 Thermal origin

Comparison between the stochastic and deterministic profile of total power demand (see Fig.4a) shows both similarities as differences: The difference between the deterministic and stochastic load profiles averages nearly zero determining that the total load for both the appliances and heating is similar. Differently from the original total load in Fig.3a, the standard deviation σ^+ and 3σ overall become high (see Fig.4a) in the order of 1 and 2 kW respectively. The high deviations σ^+ and 3σ denote that - although the total energy demand remains similar- the time shift and the number of peaks in heat load between a deterministic and stochastic profile is significant

Due to shorter intervals without heating in the stochastic profile compared to the deterministic, a 1 to 2 K higher average operative indoor temperature is noticed in the stochastic approach (see Fig.4b). The high σ and 3σ are due to comparison between non-heated and heated moments as the moments of heating are not the same in the different situations.

3.3.2 Electric effect

Similar to results concerning the thermal load, as the average electric loads are alike between the deterministic and stochastic approach (see Fig.4a), also the *aver-*

age electric response of an example grid (see appendix B) by means of a voltage drop shows small deviations (see Fig.4c).

Although the average response remains the same, deviations σ and 3σ of up to 0.5 and 1.5 V respectively are found, (see Fig.4c) with a long tail in the opposite direction as may be noticed in Fig.3c. As mentioned earlier, the effect of peak loads accumulates at grid-level. In real life and in the stochastic approach of occupant behaviour not all peaks occur at the same time and a peak load not only influences the observed voltage at the respective dwelling but also the observed voltage at the adjacent dwellings between the respective dwelling and the grid source, resulting in high σ and 3σ . The long tail in the opposite direction as Fig.3c is caused by the deterministic profile where - in contradiction to the stochastic profile - all dwellings require heat at exactly the same moment resulting in strong grid loads.

4 Conclusions

An integrated approach of probabilistic occupant behaviour in buildings with a physical multi-domain impact has been modelled and presented. The human behaviour considering occupancy, the use of lighting and the use of electric appliances in dwellings has been implemented and is used for simulation of coupled thermal and electrical systems in the building stock. Implementing stochastic occupant behaviour influences the internal heat gains which in turn influences the heat load of the building and the switch-on and -off moment of e.g. an electric heat pump. This, together with the power demand of the used electric appliances and possible on-site generation determine the load on the electric grid and possible instabilities.

By means comparison for a reference building zone and example grid, the importance of stochastic modelling of occupant behaviour in dwellings at both the building and district scale. Comparison between a deterministic approach as proposed in ISO 13790 and the use stochastic profiles shows that the direct *first order* effect is *on average* rather small: the difference in total internal gains and its influence on the indoor temperature averages nearly zero and the standard deviations σ are small, however high peaks may occur. Also the difference in effect on the electric distribution grid voltage averages nearly zero, however here strong peaks occur which are of most importance for the grid stability. When taking in account the *second order* effect of heating by means of electricity, much larger differ-

ences are noticed: due to longer and more differentiated occupancy times, the average indoor temperature rises. Furthermore, the moment of heating differentiates compared to a deterministic approach resulting in more but smaller peak demands towards the electricity grid.

5 Acknowledgements

The authors gratefully acknowledge the K.U.Leuven Energy Institute (EI) for funding this research through granting the project entitled *Optimized energy networks for buildings*.

A Building model

A high-order lumped capacitance model for predicting the unsteady building response is developed within Modelica [5, 27]. Solar radiation absorbed by the exterior surface is implemented based on the incident solar irradiation as found by the calculations in the External package depending on time, inclination and orientations and the short-wave absorption coefficient of the surface. Long wave radiation between the surface and environment is determined based on the retrieved sky temperature. The convective gains and the resulting change in air temperature of a thermal zone are modelled as a thermal circuit based on convective heat exchange with the walls and ventilation. Similar to the model for a wall, a thermal circuit formulation for the direct radiant exchange between surfaces can be derived. The heat exchange by long-wave radiation is simplified by means of a delta-star transformation [13] and definition of a radiant star node. The diffuse solar gains are divided based on the surface and emissivity of all surfaces. For direct solar gains, a factor for each surface can be given as parameter of the zone, to which direct solar gains will be divided. The star node formulation of both convective and radiative heat exchange in a thermal zone allows a straight-forward formulation for the influence of internal gains by adding them in the heat balance of radiant star node and air node of the zone of interest. Comparative literature [16] shows that the made simplifications remain a high accuracy.

The thermal model of a window is similar to the model of an exterior wall but includes the absorption of solar irradiation by the different glass panes and the transmission to the adjacent indoor zone. The properties for absorption by and transmission through the glazing are taken into account depending on the angle of incidence of solar irradiation and are based on the

output of the WINDOW 4.0 software [8] as validated by Arasteh [1] and Furler [9]. The transmitted diffuse short-wave solar radiation is treated to strike all room surfaces weighted to their surface and emissivity, whereas the direct short-wave solar gains are modeled to fall mainly on the floor. However, only small differences would arise when making different assumptions on the distribution of the transmitted energy [16].

A meteo model is implemented in the simulation environment and data are derived from Meteororm 6.1 for Uccle, Belgium. The zenith angle of an inclined surface is calculated internally [11] whereby the anisotropic sky dome model presented by Skartveit and Olseth [24, 6] is implemented.

The ideal heating system consists of an unlimited convective power controlled by a set point of 21 degrees Celcius for the operative temperature when occupancy occurs.

The reference building zone measures 36 m^2 by 3 m representing the dayzone of an average house. The construction consists of a highly-insulated cavity wall with 20 cm of mineral wool and a large south-facing double-pane window. For the stochastic determination of user behaviour, 1 to 5 occupants have been implemented respectively, an array of 16 bulbs has been taken into account and 10 electric appliances (i.e. a hob, a micro wave, two TV's, a vacuum cleaner, a hifi, an iron, a pc, a dishwasher and a washingmachine) are modelled.

B Grid model

The example grid for modelling grid voltage drops used within this work consists of a simplified grid based on the Modelica Standard Library 3.1, consisting of a constant voltage source of 230 V and lossy RC-lines. The RC-lines have each a capacitance of 0.1 mF/m , a resistance of 0.05 $m\Omega/m$ and have a length of 100 m . Here, 5 dwellings are coupled in parallel to the grid. The power demand of the dwelling determines the electric current influencing the grid voltage at both the dwelling and the neighbouring dwellings.

Note that the implemented model only serves to interpret the importance and impact of stochastic modelling of occupancy behaviour in coupled thermal and electric systems in buildings. Within the same project of the *K.U.Leuven Energy Institute* mentioned in the acknowledgements, a detailed grid model is developed.

References

- [1] D K Arasteh, J Hartmann, and M Rubin. Experimental verification of a model of heat transfer through windows. *ASHRAE Transactions*, 93(1):1425–1431, 1986.
- [2] ASHRAE. *2009 ASHRAE Handbook: fundamentals*. ASHRAE American Society of Heating Refrigerating and Air-Conditioning Engineers, Atlanta, 2009.
- [3] R Baetens, R De Coninck, L Helsen, and D Saelens. The impact of domestic load profiles on the potential of building integrated photovoltaic systems in extremely low-energy dwellings. In *Renewable Energy Research Conference*, pages 3–14, Trondheim, June 7-8, 2010.
- [4] R Baetens, R De Coninck, L Helsen, and D Saelens. The impact of the heat emission system on the grid-interaction of building integrated photovoltaics in low-energy dwellings. In *8th International Conference on System Simulation in Buildings*, page P137, 2010.
- [5] J Clarke. *Energy simulation in building design*. Butterworth-Heinemann, Oxford, 2nd ed. edition, 2001.
- [6] S Darula, R Kittler, and C Gueymard. Reference luminous solar constant and solar luminance for illuminance calculations. *Solar Energy*, 79(5):559–565, November 2005.
- [7] Roel De Coninck, Ruben Baetens, Bart Verbruggen, Dirk Saelens, Johan Driesen, and Lieve Helsen. Modelling and simulation of a grid connected photovoltaic heat pump system with thermal energy storage using Modelica. In Philippe Andre, Stephane Bertagnolio, and Vincent Lemort, editors, *proceedings of the 8th International Conference on System Simulation in Buildings*, Liege, 2010.
- [8] E U Finlayson, D K Arasteh, C Huizenga, M D Rubin, and M S Reilly. WINDOW 4.0: Documentation of calculation procedures, 1993.
- [9] R A Furler, P Williams, and F K Kneubühl. Experimental and theoretical studies on the energy balance of windows - NEFF Project report 177.1, 1988.
- [10] P Hoes, J Hensen, M Loomans, B Devries, and D Bourgeois. User behavior in whole building simulation. *Energy and Buildings*, 41(3):295–302, March 2009.
- [11] M Iqbal. *An introduction to solar radiation*. Academic Press Inc, New York - London, 1983.
- [12] ISO/FDIS 13790. *Energy performance of buildings - Calculation of energy use for space heating and cooling*, volume 2007. 2008.
- [13] A E Kenelly. Equivalence of triangles and stars in conducting networks. *Electrical World and Engineer*, 34:413–414, 1899.
- [14] S A Klein. TRNSYS - A transient simulation program, 1973.
- [15] A N Kolmogorov. Über die analytischen Methoden in der Wahrscheinlichkeitsrechnung. *Mathematische Annalen*, 104:415–458, 1931.
- [16] R J Liesen and C O Pedersen. An evaluation of inside surface heat balance models for cooling load calculations. *ASHRAE Transactions*, 103 Part 2:485–502, 1997.
- [17] J V Paatero and P D Lund. A model for generating household electricity load profiles. *International Journal of Energy Research*, 30(5):273–290, April 2006.
- [18] J Page, D Robinson, N Morel, and J Scartezzini. A generalised stochastic model for the simulation of occupant presence. *Energy and Buildings*, 40(2):83–98, 2008.

- [19] W Parys, D Saelens, and H Hens. Coupling of dynamic building simulation with stochastic modelling of occupant behavior in offices – a review-based integrated methodology. *Journal of Building Performance Simulation, Accepted*, 2010.
- [20] I Richardson, M Thomson, and D Infield. A high-resolution domestic building occupancy model for energy demand simulations. *Energy and Buildings*, 40(8):1560–1566, 2008.
- [21] I Richardson, M Thomson, D Infield, and C Clifford. Domestic electricity use: A high-resolution energy demand model. *Energy and Buildings*, 42(10):1878–1887, October 2010.
- [22] I Richardson, M Thomson, D Infield, and A Delahunty. Domestic lighting: A high-resolution energy demand model. *Energy and Buildings*, 41(7):781–789, 2009.
- [23] D Robinson, N Campbell, W Gaiser, K Kabel, A Lemouel, N Morel, J Page, S Stankovic, and A Stone. SUNtool – A new modelling paradigm for simulating and optimising urban sustainability. *Solar Energy*, 81(9):1196–1211, September 2007.
- [24] A Skartveit and J A Olseth. Modelling slope irradiance at high latitudes. *Solar Energy*, 36:526–541, 1986.
- [25] M Stokes, M Rylatt, and K Lomas. A simple model of domestic lighting demand. *Energy and Buildings*, 36(2):103–116, 2004.
- [26] P Strachan, G Kokogiannakis, and I Macdonald. History and development of validation with the ESP-r simulation program. *Building and Environment*, 43(4):601–609, April 2008.
- [27] C P Underwood and F W H Yik. *Modelling methods for energy in buildings*. Blackwell Publishing, 2004.
- [28] D Vanneste, P De Decker, and I Laureyssen. Woning en woonomgeving in België, 2001.
- [29] B A Wichmann and I D Hilll. Generating good pseudo-random numbers. *Computational Statistics & Data Analysis*, 51(3):1614–1622, 2006.
- [30] J Widén and E Wäckelgård. A high-resolution stochastic model of domestic activity patterns and electricity demand. *Applied Energy*, 2009.

Model Based Systems Engineering for Aircraft Systems – How does Modelica Based Tools Fit?

Ingela Lind Henric Andersson

SAAB Aeronautics

SE-581 88 Linköping, Sweden

ingela.lind@saabgroup.com henric.andersson@saabgroup.com

Abstract

Saab Aeronautics has chosen Modelica and Dymola as part of the means for model based system engineering (MBSE). This paper will point out why a considerable effort has been made to migrate models from other simulation tools to Dymola. The paper also shows how the models and tools are used, experiences gained from usage in an industrial context as well as some remaining trouble spots.

Keywords: MBSE; Dymola; Aircraft simulation; Model integration; Modelica

1 Introduction

Engineering aircraft systems is a complex task partly due to the factors; expensive equipment, expensive tests, long lead times, safety constraints, varying environmental conditions, e. g., temperature, pressure, and g-loads but also weight and space constraints, which may lead to high interaction level between systems, interaction between engineering domains, and finally, sensitivity to shortage of technically broad and experienced staff.

By using Model Based Systems Engineering (MBSE), much of the information regarding a system can be collected into an executable description, a model. This helps information sharing between people, encouraging cooperation over technical disciplines such as, fluid mechanics, electrical engineering and software engineering thereby helping the definitions of interfaces between systems and algorithm development of embedded systems. Integrating models from different disciplines into the executable model forces focus on the system boundaries. Models are also good tools to increase the in-depth understanding of a complex system and for training new staff. The most likely pay off is that by using

MBSE, problems can be detected earlier than by using document based systems engineering, where many problems may be detected when the first test aircraft has been built.

At Saab Aeronautics, several projects for increasing the use of MBSE are ongoing. A few of these projects are partly EU financed and performed in cooperation with most of European aircraft industry (Crescendo [3], Clean Sky [4]). The ambition is to:

- detect problems within a system or an engineering domain early,
- detect problems between systems and engineering domains early,
- increase the ability to optimize design for different purposes (such as total fuel consumption),
- detect ambiguous and/or conflicting requirements early,
- reduce the amount of implementation errors detected late,
- reduce project risks,
- gain better control of model variants, model fidelity and approved usage of models,
- get more effective system engineering,
- reduce testing time and cost,
- effectively use data from tests,
- get better secondary products, such as training simulators for pilots and technicians,
- give more fun for systems engineers as happy engineers perform better.

The rest of the paper is outlined as follows. Section 2 contains a description of the model based process and different aspects of models and Modelica usage within this process. A deeper discussion on models and tools integration is presented in section 3, section 4 reports the potential future needs and section 5 concludes the paper.

2 Model Based Development

The model based development process can be described from many different viewpoints, see e. g. [8]. Here, the viewpoint from a systems engineer specialized in systems modeling and simulation is taken. In Figure 1, an overview of the process is given. The basic idea is that as much as possible of system functional and nonfunctional aspects should be tested as completely and cheap as possible to find system design and implementation errors as close to their origin as possible. Each engineer takes responsibility that his or her work is well done and checked. The practice follows typically ARP4754 [5], but the NASA Standard 7009 [7] give many useful ideas. Another driving factor is that MBSE supports multidisciplinary optimization, a fact utilized in research projects as CleanSky [4]. To make MBSE possible, a set of tools is needed. The applications for these

tools cover many aspects of the engineering tasks; from requirement tracking, construction, configuration management, specification, modeling, simulation, report writing, archiving, project planning etc. There are many technical disciplines involved, most with specific specialist tools. This paper mostly focuses on the technical discipline of fluid dynamics with embedded hardware and software, but might be applicable to any mechatronic system.

The major reason to migrate from previous generation of simulation tools, which was made for simulation only, to Dymola is that Dymola supports all the mandatory aspects of tool integration, as described below. The choice of Dymola in front of other Modelica tools that also fulfills the technical requirements depends mostly on two aspects. It is owned by a large tool vendor and can thereby be trusted to live long enough (10-30 years) and there are consultants available that speaks Swedish.

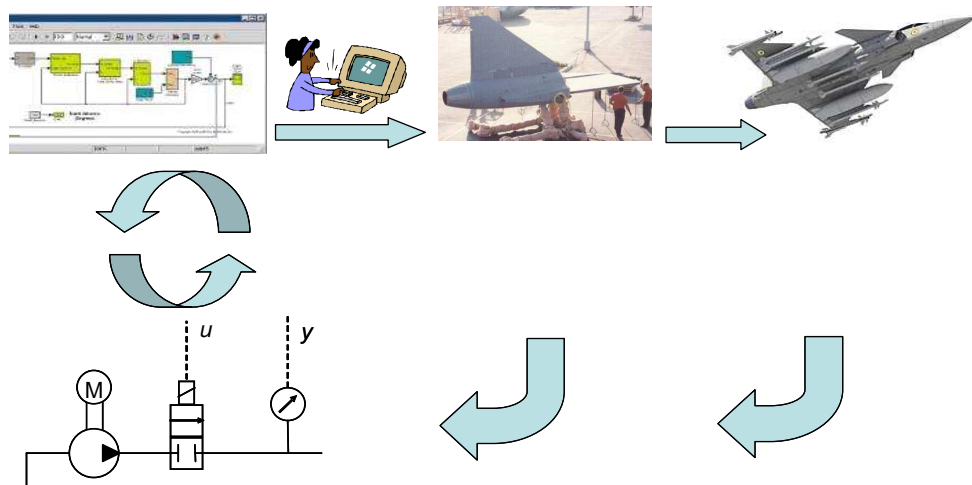


Figure 1 Model based development process. The first loop is small and fast and involves desk top simulation of one model, either control software specification or the model of the physical part, with the other model run as hosted simulation. The second loop demands more work to close the loop. Code should be made for the target computer, code for the physical part of the system might need to be exported to the simulator platform, and/or the physical parts of the rig or simulator prepared. To feed back results to the models good comparison and tuning facilities are needed. The third loop involves the airplane, which means typically expensive tests. For successful feedback, measurement, comparison and tuning facilities are needed.

2.1 Typical Parts in an Aircraft System

Aircraft systems typically involve three major types of parts; equipment, avionics with embedded software and surrounding. Equipment is things like gear boxes, valves and pipes, batteries, sensors, heat exchangers, reservoirs, etc. Avionics is computer hardware and software which fulfills requirements for aviation use. The surrounding is

everything from connected aircraft systems, ambient conditions to pilot commands.

2.2 Modeling systems of physical equipment

First, the physical part of the system is modeled, from first principles using supplier data, bench test data, previous experience, geometrical data, and sometimes also results from continuous fluid dynamics computations. To make the modeling effective, it is important to have a library of mod-

eling components built for actual purpose and in an appropriate level of detail. For environmental control systems and fuel systems, Saab has together with Modelon AB developed libraries based on the stream construct in Modelica. These are commercially available [1]. It is also important to have a good tool for tuning parameters of model components to fit well with available data. Identifiability, which means that there exists a unique parameterization of the model or solution to the optimization problem of fitting the model output to measurement data [2], is often an issue, since physically parameterized components often have several ways to affect the same property. It can be hard to determine which parameters that are affected by identifiability problems for the identification problem at hand, so any means to support this task would be welcome.

The model is (partly) reused in several different environments on different platforms with and without real time requirements and with different interfaces. This requires special care in designing the model architecture and the libraries. The Modelica construct of replaceable classes is useful for making desktop simulations faster when parts of the model can be switched off before translation, which implies that whole sections of equations don't turn up as compiled code. A typical example is the model of a fuel drop tank that can be either present and attached or not present. We miss the possibility to do the same type of switch in the generated code, so that the choice of variant can be done after translation, preferably in run time, but without the code related to those sections of equations always slowing down simulation, see further discussion in Section 3.3.

It is also useful to have model switches that can turn off a complete section of the model, as the slow temperature dynamics in the fuel system model, which makes it easier to reach real time performance for the model without rebuilding it using another component library.

As the model is reused for many different purposes, the concept of power ports inherent in Modelica is essential. The higher abstraction level used for implementing models with non-fixed causality compared to other languages means that modeling effort can be used elsewhere.

It is important to build all components and the model such that an analytical Jacobian can be created. The reason is that the models for aircraft fluid systems tend to be large with several hundred time-continuous states, to include nonlinear

equation systems, and tend also to be stiff. Our experience is that a completely analytical Jacobian decreases simulation time compared to a partly numerical Jacobian with at least a factor 5-10, but that still means a single typical simulation run takes between 5 and 30 minutes. We have also experienced a tendency that the solvers fail more easily if the analytical Jacobian is not available.

Using the model for the system of physical equipment, tasks as first concept validation, equipment sizing, sensitivity analysis, and performance estimation can be performed. But the involved systems are so complex that it is not sufficient to base further design decisions on these results. The control software is needed to e.g. close the loop.

2.3 Software specification

The control software is used to make sure that the system reaches its control goal and to perform safety functions such as functional monitoring (FM), redundancy management (RM), and built-in test (BIT), see further [9]. Physical limitations often make it necessary to introduce new control actions. An executable model of the control software is developed in a modeling and simulation tool. In vehicle systems design at Saab Aeronautics, Mathworks Simulink and Stateflow is used together with UML tools to develop software. The tools are used to build an executable specification of the code.

Depending on conditions such as criticality assessments, target avionics, review process, tool integration, and license model, code for target is either auto generated from the tool or hand coded using the model as specification, as reported in [12]. As all information about requirements tracing, purpose and descriptions are included in the model, it is possible to automatically generate parts of the software documentation from the software models.

2.4 Close the loop

By closing the loop by hosted simulation both in Dymola and in Simulink several tasks can be performed, as further described in [10]. By using the FMI standard [6] when generating code for the hosted part an efficient handling of the connected models is achieved. It might seem as double work to make a closed loop environment in two tools, but this means that engineers can perform their

tasks respectively in the tool which they are most acquainted with and which is most suited to do the tests and changes their task depends on. The closed loop simulation is useful for rapid prototyping both for the physical part of the system and for the control software. It supports safety assessment of the system and can be used to give input to computation of static and fatigue loads as well as for performance evaluation and detailed design of the system.

2.5 Large scale simulators

An aircraft consist of a large number of interacting systems. There are needs ranging from early interaction tests between systems to training of pilots, technicians and maintenance staff that can be met using large scale simulators. The large scale simulators involve many of the aircraft systems to a varying level of detail. Some large scale simulators include target avionics (that is, are partly hardware rigs) with real time performance constraints while others are completely software based without real time constraints. Control software can either be included as a model or as target code. Equipment models are needed, but often simplified models are sufficient for the use. The highest demands on accuracy on fluid mechanical system models in this context often comes from usage in training simulator for technicians and maintenance staff or from development tasks where the interaction between several complex systems is investigated. An interesting aspect of simulator models is that correct behavior when system faults occur is required. This means that all sensors and actuators need to have several different types of faulty behavior implemented [11].

2.6 System test rigs

The aircraft systems for environmental control and fuel management are so complex that system test rigs are necessary. The system test rigs are used to test that avionics and equipment have correct electrical interfaces, that all equipment has correct mechanical interface and that pressure drops, and other functional characteristics live up to the given specifications and work well together. Depending on the flight criticality class of the system and status of the system test rig, the system test rig can also be used for flight safety checks, often in combination with ground tests in the aircraft. As the avionics is also used in large-scale simulators where the equipment is not available,

the equipment model must have an interface compliant with the real equipment.

The closed loop simulation can be used to run test before they are run in the system test rig to make sure the test will give the information needed. The results from the system test rig should be fed back into the equipment and software specification models, to improve confidence and quality.

2.7 Ground and flight tests

Ground and flight tests are used primarily to ascertain flight worthiness and for validation of requirements on aircraft level. As the use of MBSE increases we see an increased usage of ground and flight tests in order to get measurement data for model validation. At the same time, the need for ground and flight tests decreases, partly due to that more validation can be done using the model based techniques.

To use model based techniques to support the certification of an aircraft, that is, ascertain flight worthiness and validate all requirements from authorities such as EASA, is partly treated in the ongoing research program Crescendo [3].

2.8 Feedback data to models

This task needs careful planning and consideration of many aspects. First, the placements of sensors in the aircraft need to be decided upon several years to several months before use. Sensor placement is expensive to change with long lead times due to the mechanical, electrical and installation work needed, if the optimal placement is not reached at the first try. Sensors are subject to a constant revision and trade off between usability versus weight and signal storage availability, which means planned sensors are easily removed, but not easily reinserted. This makes measurement data scarce.

Most flights are non-informative from a model validation perspective. The informative flights in the outskirts of the normal envelop can be hard to reach due to weather and climate conditions, safety constraints and complex conditions to be fulfilled to reach a given state. One example is ice build up in heat exchangers, which happens in some weather and flight conditions but can be close to impossible to achieve in a dedicated flight test.

To find informative data sets in the ground and flight test measurement data base is a task that requires experience and special care to make sure that the data set is fit for use. It is often less complicated to use data from a dedicated test since the control of all conditions can be monitored with the validation task in mind.

When appropriate data sets are found these should be compared with model output. The easy part of this is to simulate the model using the same ambient conditions and the same inputs from the surrounding systems. As the complete signal interface between equipment and avionics (up to 200 signals for a single system) is not subject to monitoring during ground or flight tests it is not possible to fully provide the same conditions for the equipment model as for the real equipment. The output can be compared to the measurements to give an idea of the model quality with regard to the measured quantity. To generalize the estimate of the model quality to other quantities of interest in the complex nonlinear models is more difficult. We find the tool support at the moment not sufficient for our needs. There is ongoing research on what validation methods are industrially useful for that purpose e. g. within the Swedish National Aviation Engineering Programme (NFFP), [13].

For tuning of model parameters to make the model better fit the measurement data the Dymola Model Calibration Tool has been useful to some degree, even if a larger variety of system identification methods and better control and possibility to select optimization objectives and optimization methods is desired. When using identification, different methods sometimes give different results and which method is preferred depends on the application and the validation result.

3 Integration of models and tools

In simulation of an aircraft or any other complex product, the total model is usually composed of several sub models to be manageable. The aircraft model architecture is created and maintained in order to get explicit and clear model interfaces. It is convenient to map the “model breakdown structure” onto the breakdown structure of the represented product so that e.g. system interface definitions and responsibility allocation can be reused more easily. An example of several simulation

models connected to form a larger integrated model is shown in Figure 2.

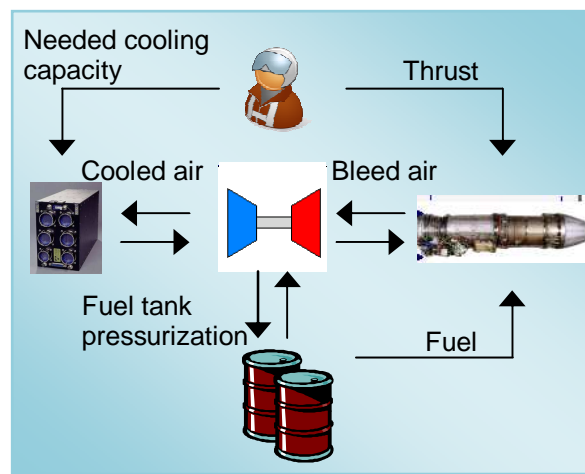


Figure 2. This is a simplified view of an integrated simulation model consisting of models for Engine, Fuel system, Environmental control system, Avionics and a Pilot model.

A smaller set of aircraft subsystem models may today be integrated in Dymola, but for larger sets of models some more specialized and powerful integration framework is still needed.

3.1 Configuration handling

Several aspects regarding configuration of models has to be handled. A simulation model is a representation of a (specified or built) aircraft system which itself is under configuration control in e.g. a Product Data Management (PDM) system. A PDM system is used for e.g. structuring, storage, change and validity control of product data related to delivery and maintenance of the products. Equipment data such as specifications, change requests of parts or documentation for certification is mature in these systems. Simulation models that represent systems/parts of the product are normally not kept within the PDM system. Information such as specification of interfaces and equations, model change requests or status accounting of models is not part of the traditional PDM scope.

Software Configuration Management (SCM) systems are suited for code management and as the simulation models are code in some format, the support for model management on code level (e.g. revision and release management) is best supported by a SCM system. Dymola supports version control of models using e.g. the SCM tool subversion (SVN).

The division of data between the PDM and SCM domains is unfortunate for many reasons, not only storage and control of simulation models. It is especially problematic in the vehicle systems modeling domain though, because data about the systems and equipment to be represented in the models are basically handled in PDM, but the model software “belongs” to SCM. So this is one of the challenges for the future to solve.

3.2 Variability

Two concepts in variability and configuration handling are variants and versions. A variant is “an option of an item which customers can choose”. Versions are sequential revisions replacing each other. Variants exist in parallel, while versions exist in a series. If an error occurs in an item it needs to be revised, and a new version is created.

As aircrafts are developed and maintained for a long period of time, the systems usually exist in several variants and versions. Also the models exist in variants and versions, but there is usually not a straight-on mapping between the PDM development tree and the model development tree. Rigor tracing has to be maintained for models used for verification, certification, and training purposes.

Models are often parameterized, meaning that one model can be used to represent a set of aircraft system variants/versions. This implies that both the parametric model (interfaces, equations, algorithms etc.) and all the parameter sets need to be under configuration control.

One driving force for parametric models or other kinds of variability is to enable reuse. In aircraft simulation it is of major importance to reuse existing models (if possible) because the verification effort for each single model drives time and cost. The number of variants should therefore be minimized, but there are situations where the requirements are incompatible and model variants are unavoidable, such as:

- different level of fidelity
- customer specific equipment models
- security (e.g. IPR)

3.3 Binding time

Binding time describes when a variable model part or function is to be bound, i.e. selected to become a mandatory part of a simulation model instance. Possible binding times include model time (also referred to as “design time”), translation time, compile time, link time, load time, and run time.

Choosing different binding time for elements affects their properties. For example, deciding to bind two model components during translation time will yield different system properties than deciding to bind these two components at run time [14].

Example of a setting is whether the simulation is to represent a single seater or a dual seater aircraft. Another example was given in Section 2.2. It is however not sure that all models with this feature as a possible choice use the same binding time as mechanism for this setting. One model may have a translation time alternatives while another uses run time switch for the same variation.

Run time binding provides in general shorter turnaround time when shifting feature. There are situations when run time binding not is sufficient, for example when propriety models are integrated in a training simulator and delivered to a customer. In this case only functionality relevant for that customer is allowed to be present in that model variant. Specific customer oriented model variants are maintained and binding is done in e.g. model time.

3.4 Integration methods

There are different kinds of integration. One is integration of software- and hardware models to form closed loop simulation. This can be performed using the hosted simulation method with Dymola or Simulink as hosting environment as mentioned above.

To integrate models into a complete aircraft simulation require models from different domains often including legacy models developed with older methods and with different software generations. A large portion of the software used for simulation in the aircraft industry is for example still implemented in different versions of FORTRAN. Some suppliers of equipment also provide

their existing models to the aircraft integrator for functional- and integration verification.

To enable integration of a wider variety of model formats, an integration framework is needed. At Saab Aeronautics there are different simulation integration frameworks based on the ADA and FORTRAN languages respectively. To be able to connect models implemented in different languages one used method is to use adaptors (or wrappers). Figure 3 shows an example of architecture for model integration.

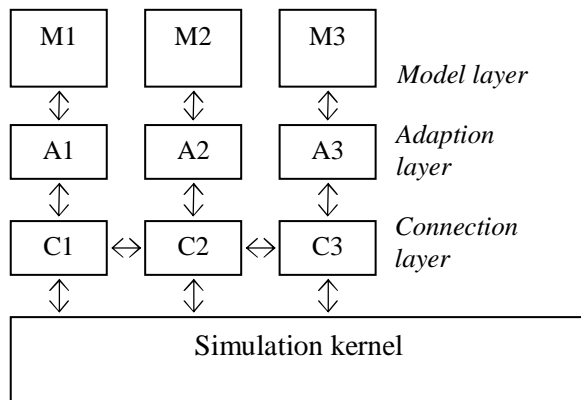


Figure 3. Example of an architecture for model integration with defined layers. The model layer is the actual simulation code in some for the framework accepted language (e.g. FORTRAN, C and/or ADA). Purpose of the connector layer is to connect models with each other and with the simulation kernel, and the adaption layer is responsible for the software language adaption (e.g. to connect C with ADA).

The emerging FMI standard [6] for model integration has the C language as a basis for software integration, which fits well with simulation code generated from Dymola. The standard also provides interface specifications in XML, which gives a powerful tool to support integration in other frameworks as it is fairly easy to map one XML scheme to another. A Modelica model created with Dymola is thereby easily integrated in a simulator with respect to the signal interface definition.

4 Future needs

Apart from the needs expressed in Section 2, the following issues are things we would like to do but have not yet found solutions to or the time to learn existing solutions.

There is a need to simulate several large system models together in a desktop environment. Each of the system models is close in size to what Dymola today handles with ease and result files get close to Windows limitations on file sizes if important signals are chosen to be stored. We do have possibility to use clusters for computation, but we lack tool support for distributing computations and results.

A related issue is support for effective code generation for multi-thread and multi-processor platforms with real-time operating systems to make complex models run able in real time.

Simple set up of batch simulations as complicated parameter sweeps or running the same simulation with several different models with good control of result files and connected with report generation is missing. Parts of it is possible to script in Modelica but especially support for detailed plot layouts and generation to reports has not yet been solved to our knowledge.

We are not yet satisfied with the auto generation of model descriptions, but this is partly due to too small amount of invested time. We would like to include more information than seem possible to do at the moment, but that might depend on that we have not yet understood how to do it.

As references for the scripting and generating capabilities related to batch simulation, plot- and document generation we have the tools MATRIXx and Matlab. These tools are according to our experience more mature in the respect of e.g. pre- and post-processing of data. There are also usable alternatives such as the python programming language [15] for data processing and spreadsheet applications for plotting.

5 Conclusions

In this paper the driver for and experiences made when shifting to Modelica as a modeling language for vehicle systems simulation at Saab Aeronautics are presented. The benefits of model based engineering are e.g. deeper insight of the systems behavior and performance as well as earlier detection of errors as compared to document based systems engineering.

For vehicle systems simulation the introduction of Modelica has largely been positive. This said there are several areas where method and tool support must be improved before MBSE and Modelica/Dymola will be the natural method to apply in all projects developing complex vehicle systems.

Areas of needed improvements are to our experience:

- better support for large size models
- support for model uncertainty and quality tracking
- support for validation of complex models using measurement data
- scripting language features for set-up, execution and post-processing of batch simulations
- better performance of code generated for real-time simulation
- code generation support for multi-thread and multi-processor targets
- generation of model documentation adapted to industry/aerospace standard

As with all methods/tools/processes there are potential for improvement, but for one of the most important ambitions we believe we have come closer to the goal; to gain happier engineers with insight and a feeling of control of their problem solving efforts.

Acknowledgments

The development methods reported in this paper has been partly inspired by and based on research funded by the research projects The Swedish Governments Agency VINNOVA's NFFP 2006-02705, 2009-01359 and 2010-01262; VINNOVA's 2007-010019 and The European Community's Seventh Framework Programmes (FP7/2007-2013) CRESCENDO (grant agreement n°234344) and JTI CleanSky.

References

- [1] Modelon AB website: www.modelon.se
- [2] Ljung, L.: System Identification, Theory for the User, 2nd edition, Prentice Hall, 1999
- [3] Crescendo website to appear at: <http://www.crescendo-fp7.eu>

- [4] CleanSky website: http://www.cleansky.eu/index.php?arbo_id=83&set_language=en
- [5] SAE Aerospace: Aerospace Recommended Practice ARP4754a, Guidance for Development, Validation and Verification of Aircraft Systems, 2008
- [6] FMI website <http://functional-mockup-interface.org>
- [7] NASA-STD-7009: Standard for Models and Simulations, National Aeronautics and Space Administration, Washington, DC 20546-0001, 2008. http://www.everyspec.com/NASA/NASA++NASA-STD/NASA-STD-7009_16145/
- [8] Steinkellner, S., Andersson, H., Gavel, H., Krus, P.: Modeling and Simulation of Saab Gripen's vehicle systems, AIAA Modeling and Simulation Technologies Conference, Chicago, Illinois, 2009
- [9] Lantto, B., Jareland, M.: Model-Based Diagnosis Studies of Complex Fluid Mechanical Aircraft Systems, In proceedings of the 25th International Congress of the Aeronautical Sciences, Hamburg, 2006
- [10] Steinkellner, S., Andersson, H., Krus, P., Lind, I.: Hosted Simulation for Heterogeneous Aircraft System Development, In proceedings of the 26th International Congress of the Aeronautical Sciences, Anchorage, Alaska, 2008
- [11] Andersson, H.: Aircraft systems modeling - model based systems engineering in avionics design and aircraft simulation. Linköping Studies in Science and Technology, Licentiate Thesis No. 1394, ISBN 978-91-7393-692-7, Liu-Tryck Linköping 2009.
- [12] Andersson, H., Weitman, A., Ölvander, J.: Simulink as a Core Tool in Development of Next Generation Gripen, In proceedings of Nordic Matlab User Conference, Stockholm, Sweden, 2008.
- [13] National Aviation Engineering Programme at Vinnova website: <http://www.vinnova.se/en/Activities/National-Aviation-Engineering-Research-Programme>
- [14] Vranić, V., Šípka, M.: Binding Time Based Concept Instantiation in Feature Modeling. In proceedings of the 9th Inter-

national Conference on Software Reuse
(ICSR 2006), LNCS 4039, Turin, Italy,
June 2006.

- [15] Python official website
<http://wiki.python.org>

Productivity improvement tool for configuration of Modelica plant models and integration with Simulink controller models

Emerson Jacob Jeganathan, Anand Pitchaikani, Elavarasan Dharumaseelan
 LMS Emmeskay Solutions Private Limited
 20, Kannadasan Salai, T. Nagar, Chennai – 600017, INDIA
 {Emerson.Jacob, Anand.Pitchaikani, Elavarasan.Dharumaseelan}@lmsintl.com

Abstract

Engineers practicing model based system design in an automotive sub-system supplier often use Modelica for modeling physical plant models and MATLAB/Simulink® for modeling the controller. Model-in-loop (MIL) simulations are performed using the S-function generated from the chosen Modelica plant model, integrated with the appropriate controller model and then simulated in Simulink. These steps are carried out by the engineer manually for the many different plant-controller configurations available in the organization. This repetitive workflow provides significant opportunities to streamline and automate the model based development process and improve productivity.

This paper presents an in-house MATLAB® GUI tool that can be used to configure the plant, select the controller, automatically generate an integrated model with the plant and controller, and simulate the resulting model. The plant model configuration information is passed on to Dymola® (the simulation environment) using the available communication (COM or DDE) to generate the plant model S-function. This tool includes post processing capabilities such as plotting the simulation results and custom plotting of metrics that are generated post-simulation.

Keywords: Automation; MIL; model configuration; simulation management

1 Introduction

Model-in-loop (MIL) simulation in non-real time environments is used to perform verification and validation testing of the controller model by the simulation engineer or test engineer, who will be referred to as “user” throughout this paper. Significant amounts of time need to be spent on configuring the plant models appropriately in Dymola and then combining

them with matching controller models in MATLAB®.

Automation of plant model configuration reduces the burden on the user who would otherwise need to spend more time understanding and configuring the plant models. The tool discussed in this paper is used with a vehicle air-conditioning system model and its controller. The development of the plant models for both vehicle and air-conditioning system is extensively described in the previous work [1]. The way in which the plant model is packaged in Modelica helps the tool in configuring the models easily and automatically.

The tool couples two different modeling platforms and, as a result, a communication channel has to be established between them to exchange configuration information. The use of DDE Server as a communication protocol for data exchange and simulation between two simulation applications (MATLAB and Dymola) was described in [2]. The various pros and cons of using various communication protocols (DDE Server, TCP/IP) for communication between MATLAB and Dymola were discussed in depth in [3].

The in-house tool that is described in this paper has communication needs only while configuring the models and not during the simulation stage. Hence DDE Server communication offers a good solution [3] for data exchange between MATLAB and Dymola.

2 Tools

The in-house tool was developed in MATLAB® R2006b version as a graphical user interface (GUI). The plant models were developed using the Modelica language version 2.2.1. The data choices for the plant models (parameters) are made available as Microsoft® Office Excel spreadsheets for ease of use. The Dymola®-Simulink® interface is used to create the S-function of the developed plant models by the

MATLAB GUI Tool. The controller S-function and plant model S-function are integrated by the MATLAB GUI Tool.

3 Plant and Controller Models

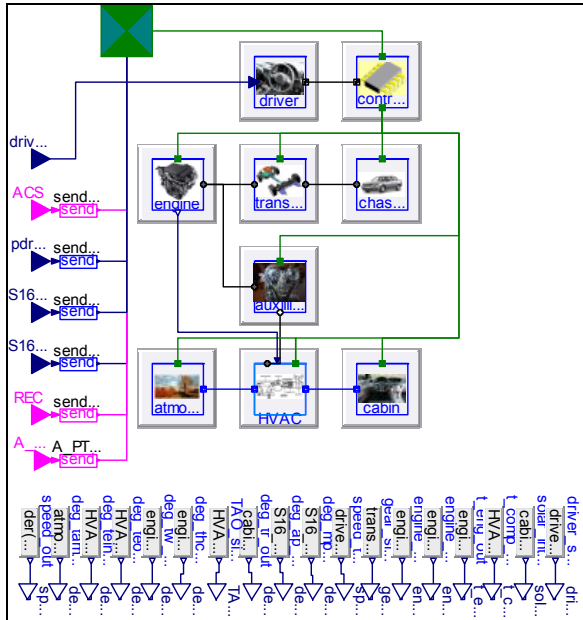


Figure 1: Plant model Architecture

The plant model architecture shown in Figure 1 consisting of the vehicle subsystems (driver, controller, engine, transmission and chassis) and the vehicle air-conditioning subsystems (compressor drive-pad, HVAC systems, cabin and atmosphere) in Modelica is made with all the main subsystems being replaceable. The variants of a particular subsystem are made by extending the same interface.

This scheme provides flexibility to choose different implementations for each subsystem as only their interface models are instantiated as replaceable in the architecture. Thus the same architecture model can be configured to represent different vehicle platforms easily.

The climate system controller model was available as a Simulink S-function. The controller exercises its control action over various components of the HVAC system to maintain the cabin at a set temperature. The controller receives sensor signals from various plant systems as feedback inputs.

A successfully integrated simulation of the developed plant models with the controller model is the main goal of the tool developed. An integrated plant and controller model is shown in Figure 2.

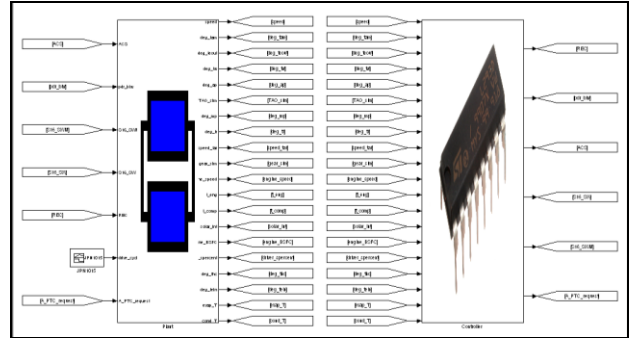


Figure 2: Integrated plant and controller model

4 Manual Process

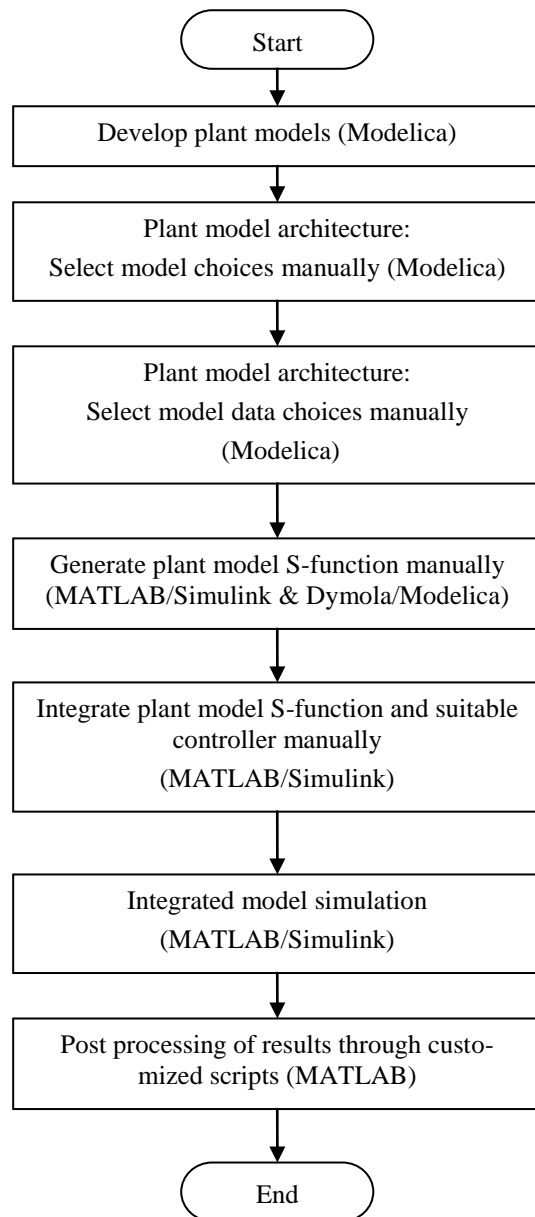


Figure 3: Flow chart - Manual process

The existing process involves configuration of the system model by applying the model choices and data choices for all top-level subsystems in a Modelica code layer by the user manually. Then the S-function is generated for the plant model using Dymola. This S-function is used in MATLAB/Simulink environment along with the controller model. The plant S-function and controller model are integrated manually and simulated in Simulink. Post-processing is done through customized m-scripts for plotting the simulation results. This process flow is depicted in a flow chart in Figure 3.

5 Modeling Guidelines

The power of object-oriented, acausal modeling language Modelica is utilized to the fullest in developing the plant models. The plant model consists primarily of components from the Modelica Standard Library (MSL). Some special needs were satisfied by creating additional custom models in Modelica as explained in [3], [4] and [5].

The plant models used in this work were done as a separate package i.e., library following certain guidelines so that the models can be interpreted and used in an easy and efficient manner by both the MATLAB GUI tool as well as the plant engineer who develops / updates the Modelica plant model.

All the top level subsystems are abstracted and individual interface models that are partial models are made. The complete plant subsystem model is created by extending the respective interface model (5 speed AT as well as 6 speed AT are extended from base class “Transmission”) and filled in with system equations. The package structure of the Modelica plant models is shown in Figure 4 and is discussed in more detail below.

5.1 Interfaces

Interfaces are partial Modelica models that form the base models for components and implementations containing just the interface details. An Interface contains the ports which interact with other models and the icon for identification.

For example, “Transmission” interface will have only two mechanical ports i.e., one for connecting to the engine and the other to the wheels. The interfaces also contain bus connectors to provide interface to the controller model.

5.2 Components

Apart from the Modelica Standard Library (MSL) components for this HVAC system simulation, the special needs were satisfied by creating additional custom models in Modelica. These are used as constituents of the top-level subsystem models.

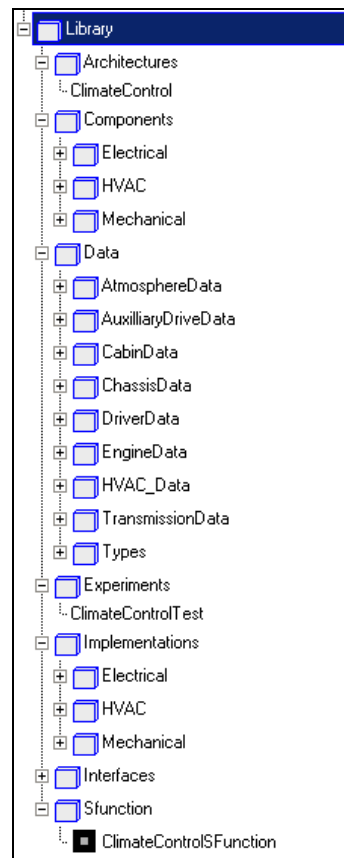


Figure 4: Library package

5.3 Implementations

Implementations are the variants of each subsystem model, which were developed extending the “interfaces” employing the “Components” described above and from the MSL library. These are complete models with respect to equations that represent the actual system.

For example “Six speed AT”, “Five speed AT”, “CVT” were variants of transmission i.e., created by extending the “Transmission” interface model. This package is parsed by the SysInit GUI tool shown in Figure 7, to populate the list of various variants available for a particular model.

The GUI bins all the implementations based on the interfaces from which they are extended. As long as new implementations are added to the package following the structure explained above, the tool will automatically add the new variant to the list of choices for that sub-system.

5.4 Data

The data for each model - the set of parameters required by the “Implementations” and “Components” is implemented using the “record” class in Modelica. For example a “Six speed transmission data” record has the gear ratios and engagement time for a six speed transmission.

To facilitate data handling by SysInit GUI tool shown in Figure 7, separate Excel sheets are created in line with the records in Modelica to allow the user to quickly edit/create the data parameters for a particular model. A sample Excel data sheet (“Six Speed AT Data”) for a “Six speed AT” model is shown in Figure 5.

Name	Description	Default Value	Type
GR1	Ratio of first gear	3.52	scalar
GR2	Ratio of second gear	2.042	scalar
GR3	Ratio of third gear	1.4	scalar
GR4	Ratio of fourth gear	1	scalar
GR5	Ratio of fifth gear	0.716	scalar
GR6	Ratio of sixth gear	0.586	scalar
J1	Inertia of first gear	1.10E-003	scalar
J2	Inertia of second gear	9.00E-004	scalar
J3	Inertia of third gear	7.00E-004	scalar
J4	Inertia of fourth gear	5.00E-004	scalar
J5	Inertia of fifth gear	3.00E-004	scalar
J6	Inertia of sixth gear	1.20E-004	scalar
FDRatio	Ratio of final drive	3.769	scalar
engagement_time	Engagement time (sec) for clutches	0.1	scalar
main_clutch_eng_time	Time taken for main clutch to engage	2	scalar
upshift_speeds_data	Gear Upshift speeds	GearUpshiftSpeeds	hyperlink
downshift_speeds_data	Gear Downshift speeds	GearDownshiftSpeeds	hyperlink
GearUpshiftSpeeds			
		989	1699
		1019	1750
			2491
			2566
			3605

Figure 5: Excel data sheet

The tool convert the data in the Excel sheet into appropriate record when writing the S-function model. Consistency in data set has to be ensured between Modelica sub-system model choice’s base record and the parameter excel file, which is uniquely identified by having its file name same as the model choice name.

5.5 Architecture

Architecture is the representative assembly of all top-level subsystem partial models with connections between them (conventional vehicle, parallel hybrid vehicle etc.). This has the interconnection of the interface models to create the skeleton of the overall vehicle system i.e., driver, engine, transmission, HVAC, cabin, etc., are connected to form the vehicle system.

The overall plant model architecture (Figure 1) in Modelica is made such that all the main subsystems are replaceable. This architecture provides flexibility to choose different implementations for each subsystem.

In this way, the same model can be configured to model different vehicle systems easily. The tool can use this architecture model as selected by the user as the base model on top of which it applies the model choices and data choices.

5.6 S-function

An “architecture” in which the sub-systems are completely defined (by re-declaring the interface models i.e., “Interfaces” with complete models i.e., “Implementations”) with the chosen model and data choices is the S-function model. The model and data choices can be reconfigured by re-declaring them. This model is written by the GUI tool and sent to Dymola for S-function generation.

An example of a Modelica model that would be compiled into an S-function is given below:

```

model ClimateControlsFunction
  "Plant S-function"
  extends Architecture.ClimateControl(
    redeclare Implementations.Mechanical.SixSpeedAT transmission(
      redeclare Data.TransmissionData.SixSpeedATData transmission_data));
  ...
end ClimateControlsFunction;
    
```

6 GUI Tool Functionalities

6.1 High level configuration

The top-level GUI which is shown in Figure 6 has various fields for selection of plant system configuration (e.g. Class A Vehicle), controller (Class A Vehicle Controller), drive cycle and fields to indicate the location of the simulation results file and summary file to be written.

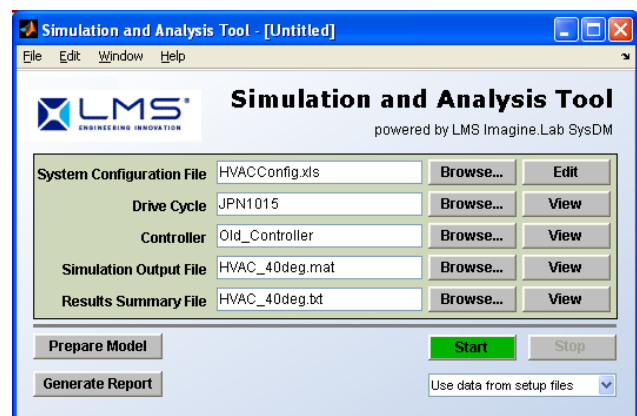


Figure 6: Top-level GUI

Using the “Browse” and “View” buttons the various choices available can be either selected or viewed respectively. The plant and controller models can be integrated and simulated using “Prepare” and “Start/Stop” buttons, while other buttons and menus provide additional functionalities such as report generation, saving the system configuration (plant and controller combination) etc.

The plant model configuration (which is stored as an Excel file) is managed by another GUI (called the SysInit GUI), which configures the plant model with appropriate sub-system model and data choices. This GUI is invoked by the “Edit” button corresponding to the System Configuration File field in the above GUI. The functionality of SysInit GUI is explained in the next section.

6.2 Plant configuration

The vehicle plant model is configured using the SysInit GUI shown in Figure 7. The SysInit GUI has three columns namely Models, Model choices and Data choices.

For each of the top-level vehicle subsystem model (say transmission), the model choices available for that model (5 speed AT, 6 speed AT, CVT etc) are displayed in the middle column and the data choices (Excel data sheets which have parameters like inertia, efficiency, gear ratio) for the chosen model choice are displayed in the last column as drop down menus.

These drop-down menus are automatically populated with model and data choices from the Modelica plant model package as explained in the best practices section.

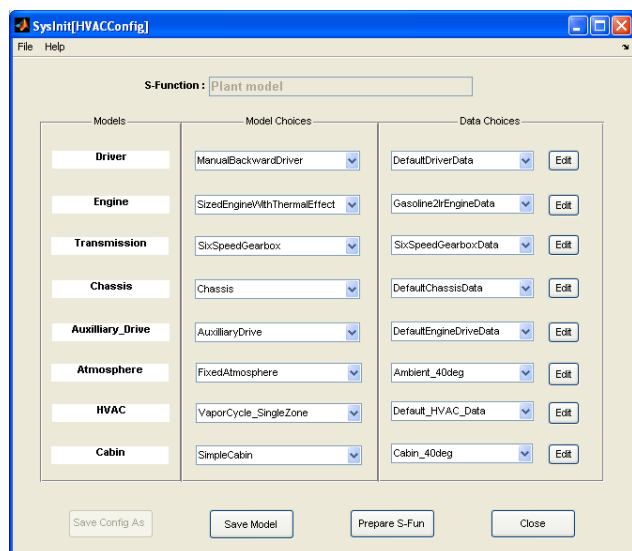


Figure 7: SysInit GUI Tool

The data for populating the model choices in the SysInit GUI is obtained by parsing the Modelica sub-system models that are developed following the “Best Practices” described previously in this paper. The data choices are obtained by parsing the Excel files which are in correspondence with the model choices.

The model and data choice displayed by default are loaded from the system configuration Excel file and can be changed as per the user’s needs. The changed configuration can once again be saved as an Excel spreadsheet with appropriate name. Once the desired choices are made for the selected plant model, a Simulink S-function can be generated using the “Prepare S-Fun” button.

The plant model configuration information is written as a Modelica model file using the architecture model specified by the user. This model is passed on to Dymola by establishing a DDE (Dynamic Data Exchange) server communication between MATLAB and Dymola to generate the plant model S-function. This simplified process enables the user to quickly create variants of plant models via different configurations of the sub-system models and parameter data choices.

The plant model Modelica file (.mo file), which may be useful for debugging can be generated using “Save Model” button, while that particular chosen system configuration can be saved to an Excel file for retrieval at a later time.

6.3 Integration with controller

After the S-function is generated for a particular plant model configuration, the top-level GUI tool integrates the chosen controller and drive cycle for simulation with the plant S-function based on matching input and output signals names in them. The drive cycle and controller library are shown in Figure 8. An integrated plant and controller model is shown in Figure 2.

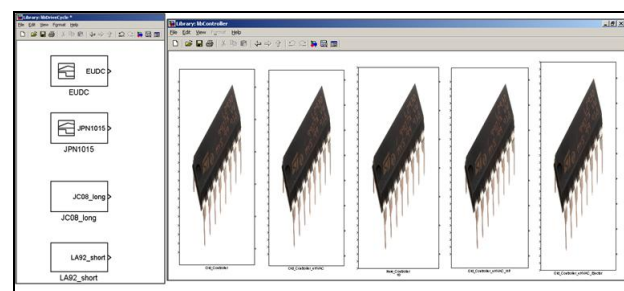


Figure 8: Drive cycle and Controller library

6.4 Simulation and post-processing

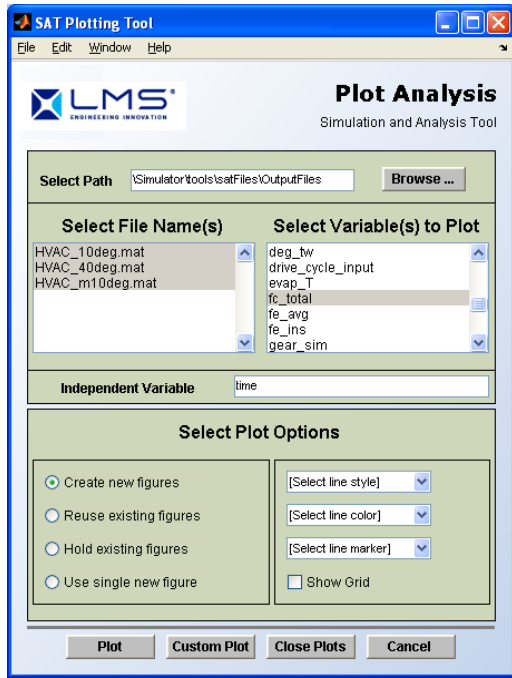


Figure 9: Generic Post Processing GUI

The integrated plant and controller model is simulated from the top-level GUI tool, which stores the result and summary in the specified files once simulation is done. Post processing of the results can be done by two plot GUI's which are shown in Figures 9 and 11.

The plot GUI shown in Figure 9 is a generic one capable of plotting same variables between multiple files and one such sample result is shown in Figure 10.

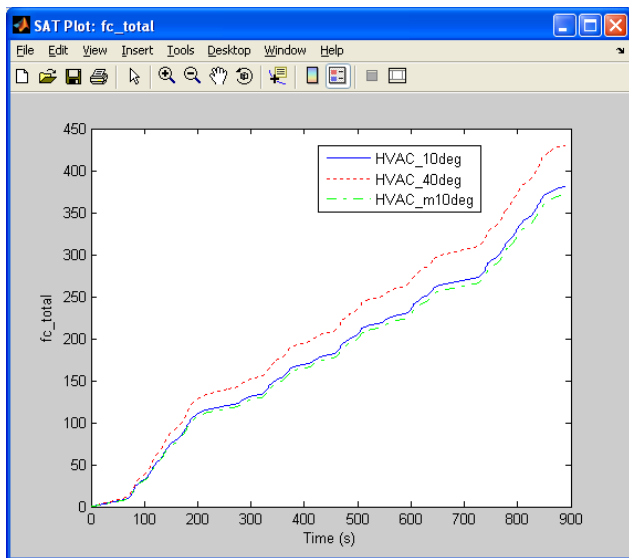


Figure 10: Sample Result (Generic Post Processing GUI)

The plot GUI shown in Figure 11, Custom Post Processing GUI is a specific one which can be used to get “metrics” i.e., some processing of results in different files to get a specific plot. This processing can be done through a MATLAB m-script and applied on the results to get the specific plot.

These post-processing methods can be quite useful for users, say in benchmarking different controllers with the same plant model.

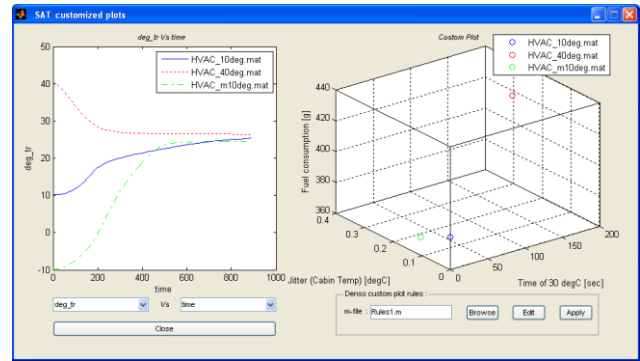


Figure 11: Custom Post Processing GUI

6.5 Flow chart

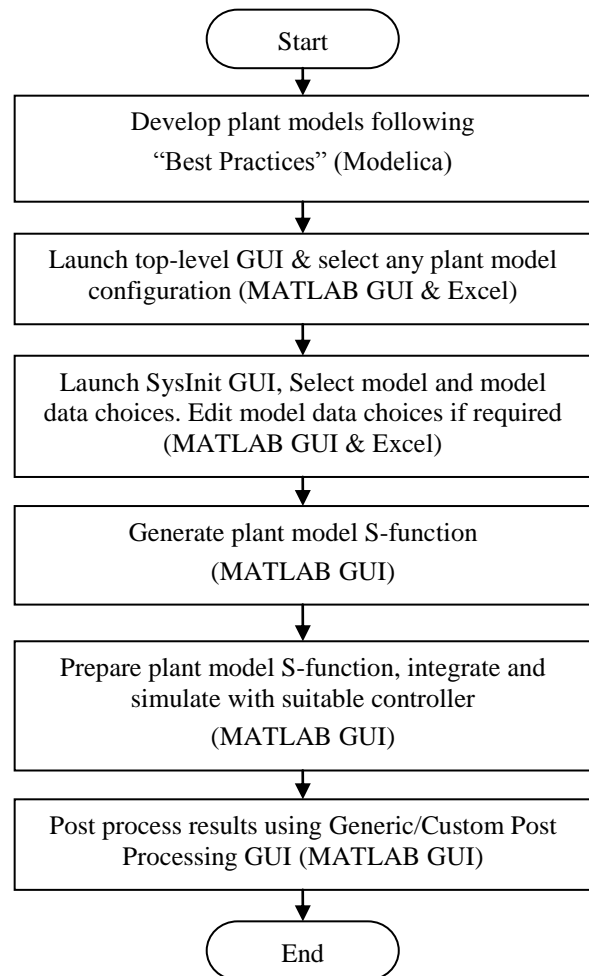


Figure 12: Flow chart – MATLAB GUI Tool process

The flow chart of the processes followed in the in-house developed MATLAB GUI Tool is shown in Figure 12.

7 Practical Implementation - Climate control

The Modelica plant models developed for usage with the MATLAB GUI Tool explained in this paper was related to Climate Controller testing using Modelica plant models [1]. Best practices explained previously were taken care while modeling the plant (vehicle with HVAC system). The climate system controller model was available as a Simulink S-function. The exercise of running the closed loop model establishes the proper compatibility achieved between the developed Modelica plant models with the chosen controller.

The closed loop simulation was very beneficial for validation of control strategies and functionality of control elements related to HVAC like heater, inlet door, etc., comparison of different control strategies and in fuel economy prediction of the vehicle.

8 Future Scope

This GUI tool is developed for a particular plant model architecture, which deals with vehicle with HVAC system simulation. This tool can be quickly reconfigured to work with any plant model architecture of interest for MIL or SIL simulations (Hybrid electric vehicle controller study, etc.). The scope of the tool can also be extended to include processor-in-loop (PIL) and hardware-in-loop (HIL) simulations.

9 Conclusions

This MATLAB based GUI tool enables the user in validating or benchmarking the controller models which are in MATLAB/Simulink against various configurations of plant models which are in Modelica for model-in-loop (MIL) simulations. The most important thing the tool achieves is that it takes the user away from the model development environment so that he remains more objective. The errors that the user can introduce while configuring the plant models in Modelica are eliminated but at a cost that the plant model engineer has to pay by sticking to the guidelines established. It was observed that the time

taken by user to configure a new plant model and get the controller tested has drastically reduced by the use of this tool. The tool helps in reducing the interactions between the user and plant engineer with respect to the plant models which helps reduce the development time.

10 Acknowledgments

This work was supported by Dr. Yasunori Yokojima (LMS Japan) and Dr. Shiva Sivashankar (LMS North America). Authors wish to thank Mr. S. A. Sundaresan (LMS Emmeskay Solutions Private Limited, INDIA) for his guidance throughout this work. Authors wish to thank Mr. Bharani Shivakumar (LMS Emmeskay Solutions Private Limited, INDIA) for improving the tool considerably.

References

- [1] Anand Pitchaikani et al, Real-time Drive Cycle Simulation of Automotive Climate Control System, pp. 839-846, Proceedings of the 7th International Modelica Conference, Como, Italy, 20-22 September 2009
- [2] S.E. Pohl and J. Ungethüm, A Simulation Management Environment for Dymola, pp. 173-176, Proceedings of the 4th International Modelica Conference, Hamburg, 7-8 March, 2005.
- [3] C. Schlegel, R. Finsterwalder, H. Olsson, Using Dymola generated C-Code in specialized Client/Server Simulation Environments, Not published, Proceedings of the 4th International Modelica Conference, Hamburg, March 7-8, 2005.
- [4] M. Tiller, "Introduction to Physical Modeling with Modelica", Kluwer Academic Publishers, ISBN 0-7923-7367-7.
- [5] Dymola. Dynamic Modeling Laboratory, Dynasim AB, Lund, Sweden, <http://www.Dynasim.se>.
- [6] Mathworks, <http://www.mathworks.com>.
- [7] Modelica, <http://www.modelica.org>.